基于 LabVIEW 平台的在线监测诊断系统多任务调度策略

孙德明　何正嘉　高　强
(西安交通大学机械工程学院,西安 710049)
E-mail:sun.198@163.com

摘　要　任务调度策略,是开发复杂多任务应用程序的关键。基于优先级的设置和执行系统的选择,结合固定时间间隔调度和事件驱动机制,提出了 LabVIEW 平台两级多任务调度策略:平台级调度策略和用户级调度策略。并以一套在线监测诊断系统为例说明了调度策略的实现过程。

关键词　LabVIEW　多任务　优先级　执行系统　事件驱动　监测诊断

文章编号 1002-8331-(2003)04-0069-03　**文献标识码** A　**中图分类号** TP273

The Method of Scheduling Multitasks in the Online Condition Monitoring and Fault Diagnosis System on LabVIEW

Sun Deming　He Zhengjia　Gao Qiang
(Department of Mechanical Engineering, Xi'an Jiaotong University, Xi'an 710049)

Abstract: The method of scheduling Multitasks is the key of developing a complex and Multitasking Application. Based on setting priorities and selecting execution systems, and by means of fixed time waited and event driven, this paper puts forwards a method of two level scheduling multitasks on LabVIEW. One level is decided by LabVIEW and the other is decided by user. As an example, this method is applied to an online condition monitoring and fault diagnosis system.

Keywords: LabVIEW, Multitasking, Priority, Execution system, Event driven, Condition monitoring and fault diagnosis

1　引言

在计算机监控系统中，通常有许多任务需要实时并行处理。比如一套在线监测诊断系统(M&DS),包含了工艺量采集处理、振动量采集处理、声光报警、异常存储、界面显示、分析诊断等多个任务。如何合理地调度这些任务,使之可靠高效地运行,是编程的一个关键点,也是一个难点。

许多编程平台,比如 Visual C++和 BorlandC++等,一般是用多线程处理技术来处理多个并行任务。这样有一些不足之处:这些平台编程环境都是文本式的,程序是按照代码编写的顺序执行的，程序员看不到多个线程究竟是如何并行处理的;程序中的多个线程之间通常要处理一些共享数据,使用一些共享资源(比如内存),这就要求程序员在编写程序的时候要特别小心,避免冲突;程序员还需要编制额外的代码来管理这些线程[1]。

在 LabVIEW 平台上,就不必担心上述问题。LabVIEW 是美国 NI(National Instruments)公司开发的主要面向计算机测控领域的虚拟仪器(Virtual Instruments,简写 VI)软件开发平台,提供图形化的编程环境。它自动管理了线程的创建、激活和挂起等复杂操作，而且其多任务并行处理过程是可视化的;它还自动管理了共享资源,避免访问冲突。程序运行时,通过周期性的扫描就绪任务队列 (Active Tasks)，根据任务的优先级(Priority)、所处的执行系统(Execution System)以及在执行系统中排列的先后顺序,决定任务的执行顺序[1]。

笔者在开发 LabVIEW 平台上的多任务应用程序过程中,根据优先级的设置和执行系统的选择,结合固定时间间隔等待和事件驱动机制,总结出了两级多任务调度策略:平台级调度策略和用户级调度策略,并在 M&DS 中得到成功的应用。

2　两级多任务调度策略

2.1　平台级任务调度

2.1.1　优先级设置

在多任务应用程序中，各个任务的实时性要求可能不同。当实时性要求较高的任务和实时性要求较低的任务都处于就绪队列等待执行时，应该保证实时性要求较高的任务首先执行。比如说,在 M&DS 中,数据采集任务要比界面显示任务实时性要求高,当某一时刻二者同时处于就绪队列,就应该保证数据采集任务首先执行。在 LabVIEW 平台上,可以通过优先级设置保证就绪任务队列中的实时性要求较高的任务首先执行。通过 VI 属性设置对话框中的 Execute 属性页可以设置 VI 优先级:

Background priority(lowest)	(低优先级)
Normal priority	(正常优先级)
Above normal priority	(较高优先级)
High priority	(高优先级)
Time-critical priority(highest)	(非常高优先级)
Subroutine priority	(子程序优先级)

Background 优先级最低,Subroutine 优先级最高。

高优先级的任务通常在低优先级任务前执行。例如,如果

基金项目:国家自然科学基金资助项目(编号:50075067);"十五"国家科技攻关计划专项经费资助项目(编号:2001BA2041305)

就绪任务队列中每个优先级的 VI 有两个，那么，执行时，Subroutine 优先级的两个 VI 会占用执行时间直到结束；然后是 Time-critical 优先级的两个 VI，它们也会独占 CPU 执行时间直到结束，依次类推。但是如果就绪任务队列中较高优先级的 VI 设置了固定时间间隔等待(用户级任务调度策略之一)且等待时间结束或者处于 I/O 等待状态，它会被挂起(移出就绪任务队列)以便较低优先级的 VI 执行，直到等待时间到了或者 I/O 操作完成了，再把它激活(插入就绪任务队列)插在较低优先级的前面继续执行[2]。

这么多优先级可以设置，但并不是在每个 LabVIEW 多任务应用程序中都要用到，在具体使用的时候应该注意以下几点：

(1)VI 默认的优先级是 Normal priority；

(2)一个 VI 调用了另外一个 VI(称为 SubVI)，如果 VI 的优先级比 SubVI 的优先级高，则 SubVI 优先级自动升级，使之与 VI 的优先级相同；否则，SubVI 的优先级不变；

(3)Subroutine 优先级 VI 可以调用 Subroutine 优先级 VI，但是不能调用其他优先级 VI；

(4)Subroutine 优先级 VI 中不能调用一些函数(比如：WAIT 函数、GPIB 函数、VISA 函数或者是对话框函数等)使 VI 挂起，因为执行系统无法控制 Subroutine 优先级 VI；

(5)在一个多任务应用程序中，不要使用太多的优先级。因为较高优先级的 VIs(多个 VI)不断进入就绪任务队列，可能会将执行时间全部占用，较低优先级的 VIs 会一直处于等待状态[2]。

2.1.2 执行系统选择

优先级的设置，为不同实时性要求的任务的可靠运行提供了可靠的保证；执行系统的选择，则为相同优先级的任务的并行处理提供了基础。

在一些复杂多任务应用程序中，数十个甚至上百个 VI，不可能都用优先级来区别它们，也就是说同一个优先级内可能有若干个 VI 处于就绪状态等待执行。它们的执行时间可能差不多，也可能差别比较大。比如说，有三个相同优先级的 VI：A、B、C，执行所需的时间分别是：10、20、100 毫秒，如果 C 排在最前面，那么 A 和 B 就要一直处于等待状态，直到 C 执行完。这种顺序执行显然不太合理。为了改变这种不合理，LabVIEW 提供了六种执行系统，每种执行系统对于 CPU 的使用具有相同的权限。这样，就可以把 A、B 放在一个执行系统，把 C 放在另外一个执行系统，A、B 和 C 就可以并行执行了。

同样，可以通过 VI 属性设置对话框中的 Execute 属性页选择 VI 的执行系统：

User interface	(用户界面)
Standard	(标准)
Instrument I/O	(仪器 I/O)
Data acquisition	(数据采集)
Other1、2	(其他 1、2)
Same as caller	(同调用者)

除了 User interface 执行系统，每个执行系统拥有并维护着自己的就绪任务队列。这些执行系统不响应用户界面操作，如果执行系统队列中的某个 VI 需要更新前面板上的控件状态，该执行系统会通知 User interface 执行系统负责完成。也就是说，即使 VI 使用了 Standard 执行系统，用户界面事件还是由 User interface 执行系统负责完成。任何用户界面事件都不会阻碍后台 VI 的运行。

在选择执行系统的时候，应注意以下几点：

(1)VI 默认的执行系统是 Same as caller，指使用与调用者相同的执行系统；

(2)各执行系统只是名字有区别，对 CPU 执行时间具有相同的使用权限；

(3)用户界面事件是由 user interface 执行系统处理的，为了 user interface 执行系统更好地完成界面刷新工作，一般不要将其他 VI 或 SubVI 在该执行系统执行；

(4)一个 VI 调用了另外一个 VI(SubVI)，如果 VI 和 SubVI 的执行系统不同，维持各自的执行系统不变。

2.2 用户级任务调度

通过优先级和执行系统的合理选择，系统决定了就绪任务队列获得 CPU 进行处理的顺序，实现了平台级的任务调度；通过固定时间间隔等待和事件驱动机制，用户则可以决定哪些任务被激活进入就绪队列，实现用户级的任务调度。

2.2.1 固定时间间隔等待

为了控制某些循环处理任务的执行频率，可以在循环内部使用 WAIT 节点设置固定时间间隔等待。比如 M&DS 中的数据采集任务，其执行时间(可以通过 VI 性能测试器测得，测试器位于 Tools<<Advanced<<Profile VIs…[3])可能只有数十毫秒，以这么快的处理速度连续采集是没有必要的，一般来说 200~1000 毫秒内采集一次就可以了。设置了时间间隔之后，采集任务开始执行时开始计时，执行完一次采集，检查等待时间是否结束，如果没有结束，采集任务就会被挂起，直到等待时间结束，采集任务又被激活，执行下一次采集。这样，就可以节省出 CPU 执行时间供其他任务使用，提高程序的执行效率。

需要说明的是，如果设置的固定时间间隔小于任务的执行时间，那么只要程序不停下来，该任务就不会被挂起[3]，处于连续运行状态。所以，在设置等待时间之前，应该先测试一下任务的执行时间，保证固定时间间隔不小于任务的执行时间。

2.2.2 事件驱动机制

LabVIEW 平台上的事件驱动是通过使用 OCCURRENCE 节点来实现的，可以灵活控制任务的激活和挂起。OCCURRENCE 节点由三部分构成：Generate Occurrence、Set Occurrence 和 Wait On Occurrence。

任务的事件驱动机制就像是用一个开关控制一只灯泡，首先由 Generate Occurrence 用来生成一个事件对象(开关)，在需要的时候由 Set Occurrence 设置事件(触发开关)，包含 Wait On Occurrence 的任务(灯泡)就会被激活。比如 M&DS 中的报警任务，在数据处理任务中检验采集的数据是否超限达到报警条件，如果到达报警条件，就设置报警事件。报警任务接收到报警事件就会被激活，执行报警功能。

3 实例应用

基于 LabVIEW 多任务调度策略的可行性和易操作性，运用上面提到的两级任务调度策略，为某钢铁公司焦化厂煤气鼓风机组成功开发了 LabVIEW 平台上的 M&DS，实现了鼓风机组振动量信号和工艺量信号监测、报警、存储、远程传输以及分析诊断等功能。系统并行处理的任务有：振动量信号采集处理、工艺量信号采集处理、状态监测(包含分析诊断)、声光报警、异

常存储、常规存储和远程传输。

3.1 平台级任务调度

(1)优先级设置——考虑到系统的实时性要求和故障诊断需求,振动量和工艺量采集处理、声光报警和报警存储任务选择 Above normal 优先级;其他任务选择 normal 优先级。这里只是使用了两个优先级,因为状态监测的优先级较低,如果再设置多一些较高的优先级,可能会导致状态监测较长时间处于等待状态,监测界面刷新较慢,让用户误以为死机。

(2)执行系统选择——选择了五个执行系统。为保证数据采集任务可靠执行,使用了两个执行系统;状态监测和声光报警虽然选择了同一个执行系统,但是优先级不同,可以保证声光报警的实时性要求;报警存储和常规存储也是如此。

具体任务采用的优先级和执行系统参见表 1。

表 1 平台级任务调度表

任务	工艺量	振动量	状态	声光	异常	常规	远程
调度	采集处理	采集处理	监测	报警	存储	存储	传输
优先级	Above normal	Above normal	Normal	Above normal	Above normal	Normal	Normal
执行系统	Data acquisition	Instrument I/O	Standard	Standard	Other 1	Other1	Other 2

3.2 用户级任务调度

用户级任务调度结构示意图如图 1 所示。

(1)固定时间间隔调度——除了声光报警任务。每个任务的具体间隔时间可以通过参数设置模块进行设置。有一点要注意的就是:振动量数据采集处理任务的固定时间间隔应该是所有任务的固定时间间隔当中最小的一个。因为不刷新采集的数据,其他任务的执行没有意义。

(2)事件驱动机制——声光报警任务。其中 TechAlarm 是工艺量报警事件,VibAlarm 是振动量报警事件。二者有一个发生,声光报警任务就会被激活。

所有循环任务由同一个按钮来结束执行。按钮有两个状态:按下和弹起,一般处于弹起状态。当按下时,结束所有的循环任务,为了恢复按钮的弹起状态,用了一个逻辑或操作。

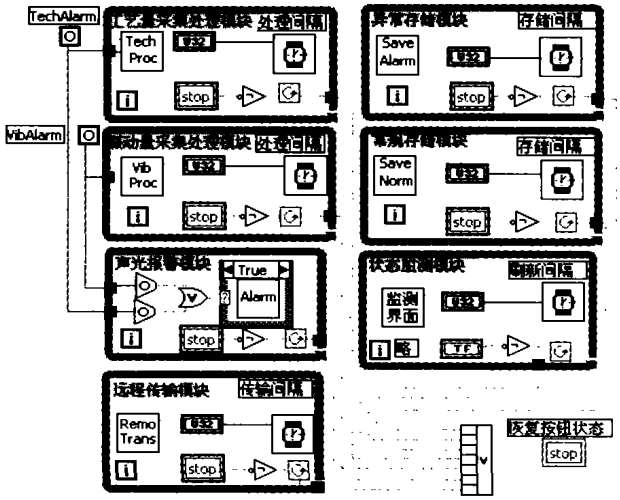

图 1 用户级任务调度结构图

4 结论

在 LabVIEW 平台上,通过用户级的任务调度,用户决定了哪些任务被激活进入就绪队列;通过平台级的任务调度,系统决定了就绪任务队列获得 CPU 进行处理的顺序。这样就确保了复杂多任务应用程序中各个任务的正常执行,大大提高了 CPU 的利用率。(收稿日期:2002 年 7 月)

参考文献

1.Using LabVIEW to Create Multithreaded VIs for Maximum Performance and Reliability.National Instrument Corp,2000

2.杨红.在 LabVIEW 上构造复杂多任务应用程序[J].测控技术,2000;(1)

3.LabVIEW Performance and Memory Management.National Instrument Corp,2000

(上接 68 页)

以上数据说明了用 MEC 求解约束优化问题的有效性与收敛性。

作者还用该文算法对多个典型约束非线性优化问题进行了仿真实验,在约束误差的允许范围内均取得了较好的结果。

6 结论

文章尝试用 MEC 求解约束优化问题。由于 MEC 具有严格的群体结构组织、完善的记忆机制、显著的首领效应,因此很便于对约束条件进行描述与度量。该文借鉴可变容差策略,定义了个体可行测度、群体可行测度、近乎可行个体等概念,利用子群体的可行测度与其优胜者的可行测度,准确描述了子群体到可行域的相对位置,并以此为指导信息引导搜索从非可行域不断向着可行域的方向进行,逐渐逼近问题的最优解。实验仿真结果证明了将 MEC 用于求解约束优化问题是非常有效的。

(收稿日期:2002 年 4 月)

参考文献

1.Chengyi Sun,Yan Sun.Mind-Evolution-Based Machine Learning:Framework and The Implementation of Optimization[C].In:Proc of IEEE Int,Conf on Intelligent Engineering Systems(INES'98),Vienna,Austria,1998:355~359

2.Jianchao Zeng,Kai Zha.An Mind Evolution Method for Solving Numerical Optimization Problems[C].In:Proc 3rd World Congress on Intelligent Control and Automation(WCICA2000),Hefei,P R Chine,Press of University of Science and Technology of China,IEEE Catalog Number:00EX393,ISBN:0-7803-5995-X,2000:126~128

3.孙承意,谢克明,程明琦.基于思维进化机器学习的框架及新进展[J].太原理工大学学报,1999;30(5):453~457

4.潘正君,康立山,陈毓屏.演化计算[M].清华大学出版社,广西科学技术出版社,1998

5.D M 希梅尔布劳.实用非线性规划[M].科学出版社,1981

6.M 阿弗里尔著.非线性规划---分析与方法[M].上海科学技术出版社,1980

7.吴志远,邵惠鹤,吴新余.基于遗传算法的退火精确罚函数非线性约束优化方法[J].控制与决策,1998;13(2):136~140