Windows NT 中的处理器调度及改进方法

范颂军 陈凡 黄厚宽 (北方交通大学计算机系 100044)

摘要:本文首先描述了 Windows NT 操作系统的调度。然后针对其不足,介绍一种新的处理器调度方法——处理器继承调度。Windows NT 操作系统若能够在今后的版本中采用此框架,将大大提高其系统的灵活性。

一、Windows NT 的优先级调度策略

1. NT 操作系统优先级调度的优点

Windows NT 操作系统是 Microsoft 公司提供的,一个不再需要 MS—DOS 支持的、崭新的操作系统。它可以很好地在当前流行的 X86 机器上运行,同时可以支持多任务运作方式,也就是能够将 CPU 时间分配给为之竞争的各个线程。对于一般用户来说,多任务的优点是可以同时打开和运行多个应用程序,而对于应用程序开发者来说,多任务的优点是能够创建使用多进程的应用程序,以及使用执行多线程的进程。

在 Windows NT 操作系统中,一个进程通常定义为程序的一个实例,是一段调入内存准备执行的一段应用程序。它可以使用 4GB 的虚地址空间(Windows NT 所使用 32 位的保护模式使用页面交换技术,容许应用程序在运行时拥有 4GB 的虚地址空间)来放有应用程序.EXE 文件的代码和数据。但进程是没有活力的,因其并未得到运行所必须的其他资源。为了让进程完成它的工作,进程必须占有一个线程;由此线程负责执行包含在进程地址空间中的代码。实际上单个进程还可以包含多个线程,而这些线程是应用程序可控制的。为了运行这些线程,Windows NT 操作系统为每个独立的线程按优先级的高低分配 CPU 时间,以轮转方式向各个线程提供时

间片，从而给人以多个线程同时运行的感觉。也就是说，多任务是由操作系统调度其进程来实现的；由调度框架决定哪个竞争线程将获得下一个时间片（每个时间片包含固定数目的时钟周期）。系统以优先级为基础安排所有的活动进程。

系统用一个两步骤过程来确定线程的基本优先级：

(1)分配给进程一个优先级类，以便系统在此进程与正在运行的其他进程相比较时决定其优先级的高低，如表 1 所示：

表 1 线程的基本优先级

线程相对优先级	空闲	普通	高	实时
时间关键	15	15	15	31
高于普通	6	9	15	26
普通	5	8	14	25
低于普通	4	7	13	24
最低	3	6	12	23
空闲	2	5	11	22
	1	1	1	16

(2)给进程拥有的线程分配相对优先级。另外系统有时会临时提高线程的优先级，也就是说每个线程还拥有一个永远不低于其基本优先级的动态优先级。事实上调度决策使用的就是动态优先级。

Windows NT 操作系统采用抢占式的调度算法。调度框架在每个时钟中断出现时获得控制：①若系统确定有一个更高优先级的线程准备运行，则系统立刻将较低优先级的线程挂起（即使它处于时间片中间），把 CPU 分配给那个拥有满时间片、具有较高优先级的线程。较高优先级的线程总是抢占较低优先级的线程，不管较低优先级的线程正在执行什么；②若当前执行线程未被抢占，则使其时间片里剩下的时钟周期记数减 1。如果减到 0，就作出下一个调度决策来决定下面将执行哪一个线程。下一个时间片总是属于那些准备执行的线程中优先级最高的一个。当一个以上的可执行线程拥有最高优先权时，时间片就给等待了最久的那个线程。

Windows NT 操作系统一般能够很好地利用系统的动态优先级调节功能，从而达到最好的整体效果：不但能够保持系统对于终端用户的高度响应性，而且充分利用 CPU 时间片。

2. 存在的问题及产生原因

由于不同的实际应用对于操作系统调度框架提出的要求也不同，只用一种调度算法并不能完美地满足所有应用的需求。

对于实时类应用，反应时间（可以用多快的速度来反应用户的是命令输入）是最重要的因素；对于批处理作业，处理结果是最重要的，时间延迟却是个小问题。硬实时（hard real - time）应用要求满足该应用特定的最迟完成时间；而对于软实时（soft real - time）应用来说，偶尔超时并非引起灾难性的后果。

Windows NT 操作系统处理实时类应用时，可以关闭掉系统正常的动态优先级调度，能够保证实现此类应用线程的优先级不会因运行时间过长而被降低，可使其一直运行在程序员指定的优先级上。通过在系统中仔细地给实时类线程分配优先级，并且保证所有的非实时类线程在一个较低的优先级运行，对于某些应用有可能得到正确的响应。尽管如此，这种方法存在着严重的、并且为大家所熟知的缺陷：若此类线程可执行程序编码有问题，例如在其中存在死循环，会导致整个系统的瘫痪；对于有些情况来说，需要使用完全不同的调度策略，以满足其特殊需要。

即使在对于通常的交互式及批处理应用处理中，Windows NT 操作系统单一的优先级调度也存在缺陷。因其不能够把一系列进程、线程作为一个整体封装起来，从而不能够相对于系统中其他部分隔离和控制它们对于处理机的使用。这将导致系统遭受攻击，出现各种不同的拒绝服务的现象，其中最常见的是在系统中出现大量的线程，它们完全强占了处理机，因此本该由系统提供的其他服务被排除在外，得不到处理。我们在测试 Windows NT Server 的性能时验证了上述情形，当处理机被强占时，相当数量的客户机所提出的服务请求得不到服务器的响应，而出现饿死（Starve Out）现象。另外，当一些完全不知其身份、完全不可信任的代码被下载，并运行在一个安全的环境下时，这种脆弱性会引起严重的麻烦。

二、一种新的处理器调度方法

由于上述情况，采用一种能支持多种调度策略的处理器调度框架的要求变得越来越迫切。在这里介绍一种

基于 Windows NT 操作系统的优先级继承调度的处理器调度方法，我们称之为处理器继承调度。

在调度模型中，与 Windows NT 操作系统一样，一个线程是一个虚拟的 CPU，其功能是来执行各种指令。在任意时刻，一个线程可以被实际分配得到一个实际的 CPU 资源，也可能没有得到；在任何时候，一个正在运行的线程的 CPU 资源可以被剥夺，并被重新分配给其他线程。

在 Windows NT 操作系统中，线程是由一些低级实体来调度的，例如在操作系统内核中的调度程序，或者是一个用户级的线程包。正好相反，线程由其他线程来调度是 CPU 继承调度的基本想法。在任何时刻，任意一个拥有实际可用 CPU 资源的线程，都可以暂时地将其分配给其他的线程，而不是自己使用此资源。这种操作很象 NT 操作系统中的优先级继承，只是它完全由特定的线程来完成，并且不涉及优先级的概念，只是 CPU 资源从一个线程到另一个的直接传输；因此得名为 CPU 继承。

一个调度线程用其大部分时间决定如何将其 CPU 资源分配给其顾客线程；其顾客线程从而继承该调度线程的部分 CPU 资源，并且把这部分资源当作它们的虚拟的 CPU。顾客线程也可以如此操作，充当调度线程的角色，在其自己的顾客线程之间分配 CPU 资源。如此往复，形成如图 1 所示的逻辑结构。

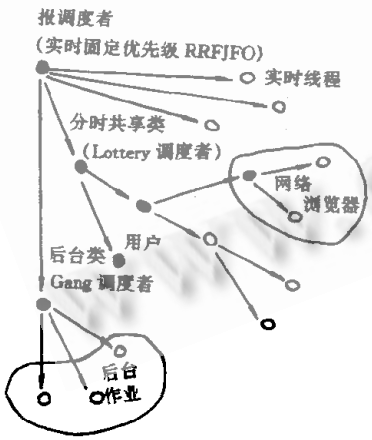

图 1

如图 1 所示，一些主要的线程可以在等待某些特定事件发生的同时，作为其他线程的调度者，把它们的处理器时间暂时地分配给某些选定的线程。接受处理器时间的线程可以进一步把它们的处理器时间分配给另外一些线程。此过程也可以不断地循环往复下去。在此系统中，只有根 scheduler 一被创建就拥有实际可用 CPU 时间；其他线程只能在分配到 CPU 时间后才能运行。对于系统中的每一个实际的 CPU 有一个根 scheduler 与其相对应；此 CPU 永远向与其相对应的根 scheduler 线程提供 CPU 时间。此根 scheduler 线程的操作决定了该 CPU 的基本调度策略。

在此调度模型中，基本调度策略可由一般的线程通过彼此合作来实现，尽管可能越过保护界限而引起安全问题，但可以通过使用完善定义的接口来解决之。例如，要实现一个固定优先级的多处理器调度策略，可以在一组 scheduler 线程（其中任一个对应一个可使用的 CPU 资源）之中，维持一个待调度的"顾客"线程的优先级化的队列。每一个 scheduler 线程顺序的选择一个线程来运行，并且在等待某些象时间片用尽（例如，一个时钟中断发生了）一样的事件发生时，会把它的 CPU 时间分配给所选择的目标线程。

如图 2 所示，假如选定的线程阻塞了，它的 scheduler 线程会得到通知，并且重新分配它的 CPU。另一方面，假如有一个不同的事件，导致该 scheduler 线程被唤醒，就会导致这个运行线程所占有的 CPU 被剥夺，并且把被立刻归还给 scheduler。其他的调度策略，象分时共享、固定优先级、rate monotonic 、fair share、以及 lottery/ stride，均可以用相同方式来实现。

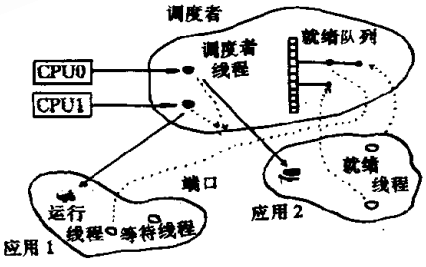

图 2

（来稿时间：1997 年 8 月）