Linux 与 VxWorks 任务调度机制分析

高鹏飞　李新明　孙　建　中国人民解放军装备指挥技术学院电子工程系(101416)

Abstract

In this paper,differences in task scheduling mechanism between the two multi-tasks operating systems Linux and VxWorks are analyzed.The two are compared with aspects of task control block,the standard of scheduling and the policy of scheduling.Further more,The implementations of task scheduling in POSIX 1003.1b specification are analyzed both in Linux and VxWorks.

Keywords:Linux,VxWorks,task scheduling,scheduling policy,priority

摘　要

分析了 Linux 和 VxWorks 两种多任务操作系统任务调度机制的异同,从任务控制块、调度的时机、调度的优先级和调度的策略方面进行了详细的分析和对比。分析了 VxWorks 和 Linux 在 POSIX1003.1b 调度标准实现上的差异。

关键词:Linux,VxWorks,任务调度,调度策略,优先级

通用的分时操作系统面向多用户的不同任务，意在追求系统整体运行的效率和资源的均衡利用，软件的执行在时间上要求并不严格。实时操作系统不同于分时操作系统,它主要是对任务进行实时的处理,要求任务的运行具有可确定性和可预测性,提供即时响应和高可靠性。由此导致通用分时系统和实时操作系统的内核在任务调度机制方面的不同。本文以 Linux(2.4 版本内核)和 VxWorks(5.4 版本)为代表,着重分析二者任务调度机制的异同。

1　进程(任务)控制块

进程是一个具有独立功能的程序对某个数据集在处理机上的执行过程和分配资源的基本单位[1]。进程是一个动态概念,具有并行特征,表现为独立性和异步性。在 Linux 中,进程和任务的概念是等价的。VxWorks 中把每个相互独立的程序称为任务,任务是 VxWorks 资源分配的单位,也是处理器调度的基本单位。

每个任务拥有各自的上下文,即拥有各自的 CPU 环境和系统资源。上下文切换时,Linux 进程上下文保存在进程控制块中(PCB)中,VxWorks 任务的上下文保存在任务控制块(TCB)中,二者的上下文内容有很大的区别。

作为通用操作系统的 Linux,其进程所使用(管理)的资源比嵌入式实时操作系统 VxWorks 要多的多，就任务控制块来说,二者的差异还是比较大的,典型的区别分析如下：

1)多用户性。Linux 是多用户操作系统,它使用用户和组标识符来控制进程对系统中文件和映像以及其他资源的访问权限。task_struct 中有四对进程和组标识符:进程 uid 和 gid、有效 uid 和 gid、文件系统 uid 和 gid、保留的 uid 和 gid。而 VxWorks 是单用户的操作系统,任务控制块结构 windTcb 中没有与用户相关联的数据结构。

2)文件打开表。Linux 的每个进程都拥有一个文件打开表,以记录该进程使用文件的情况，对应于 task_struct 中 file_struct* 类型的 files 成员。当进程退出时,内核会自动检查并关闭进程已打开但未显式调用 close( )关闭的文件。VxWorks 的整个系统共用一个文件打开表，每个任务控制块中没有记录该任务已打开的文件信息,因此当某个任务退出时,其已打开的文件自然不会被自动关闭。

3)任务之间的亲属性。Linux 的每个进程都不是孤立地存在于系统中,新的进程自创建起就置于一种表示“宗族”和“家谱”的树型组织中，初始化进程 init 是所有进程的祖先进程。在 task_struct 结构中有一组 task_struct* 类型的指针结构成员,其中 p_opptr 和 p_pptr 指向父进程,p_cptr 指向最“年轻”的子进程,p_ysptr 和 p_osptr 则分别指向其“哥哥”和“弟弟”。而 VxWorks 的任务间没有类似的亲属关系，每一个任务都是平等而独立的,即使一个任务创建并激活一个新的任务,两者之间也不存在父子关系。

另外,Linux 的进程控制块和 VxWorks 任务控制块的差别还有许多，比如 Linux 的 task_struct 中保存的进程杂凑队列的链入指针、线程信息、虚拟内存信息等。

2.1　VxWorks 的任务调度

2.1.1 VxWorks 任务调度时机

VxWorks 中任务调度的时机可分以下两种情况[2]：

1)同步任务切换,引起的原因是当前运行的任务执行下列操作:①进行阻塞、延迟、挂起的调用;②使更高优先级任务就绪而发生优先级抢占;③降低自身优先级或者退出。

2)异步任务切换,通常由中断服务使高优先级任务就绪引起。

VxWorks 的 Wind 微内核基于优先级抢占调度，并采取单一实地址空间模式,上述引发任务调度的情况发生时,VxWorks 内核立即进行任务的调度切换。

2.1.2 VxWorks 任务调度策略

多任务运行时需要一个调度算法,将 CPU 分配给就绪的任务。Wind 内核默认采用基于优先级的抢占式调度 (Priority-based preemptive scheduling) 算法，同时也可以选用轮转(Round-robin)调度算法。

1)基于优先级抢占的任务调度:指每个进程被赋予一个优先级,优先级最高的就绪进程率先执行;可抢占调度(preemptive scheduling) 是指允许将逻辑上可运行的进程暂时挂起的策略[3]。VxWorks 采用基于优先级抢占的调度算法,系统中的每个任务都拥有一个优先级,内核将 CPU 分配给处于就绪状态的优先级最高的任务。如果系统内核一旦发现有一个优先级比当前任务的优先级高的任务转变到就绪状态，内核立即保存当前任务的上下文，当前任务状态切换到就绪状态并按优先级插入到相应任务队列的队尾，然后内核切换到这个高优先级任务的上下文中执行。

Wind 内核有 256 个优先级，编号是 0~255,0 的优先级最

高,255 最低。任务的优先级可以在创建时设定,系统默认的任务优先级为 100。 VxWorks 允许任务动态改变自己的优先级,当任务执行时,它可以调用 taskPrioritySet( )改变自己的优先级。

2)轮转调度 :VxWorks 中,基于优先级抢占的调度可以与轮转调度相结合。轮转调度算法试图让优先级相同的、处于就绪状态的任务公平地分享使用 CPU 资源。如果不使用轮转调度,当系统中存在多个相同优先级的任务时,第一个获得 CPU 的任务将会独占 CPU,如果没有阻塞或其它情况出现,其它相同优先级的任务将得不到运行的机会。

使用轮转调度时,每个任务被分配一个时间段,称作它的时间片(quantum),即该任务允许运行的时间。在 VxWorks 系统中,可以调用函数 kernelTimeSlice()来选用轮转调度策略,其参数是时间片的长度,该时间片是每一个任务在放弃 CPU 给另一个相同优先级任务之前,系统允许它运行的时间长度。如果任务在它的时间片中被高优先级的任务抢占,调度器保存它的运行时间计数器,当它再一次符合执行条件的时候,调度器恢复运行时间计数器。

2.1.3 抢占上锁与中断上锁

由于所有 VxWorks 任务共存于单一的线性地址空间,当多个任务共享全局的数据结构时,需要提供对临界区的互斥访问机制。使用信号量对资源上锁是一种比较通用的互斥手段,如 POSIX 有名信号灯或无名信号灯和 System Ⅴ的信号量集。VxWorks 提供了三种类型的信号量 :二进制信号量、互斥信号量和计数器信号量。此外,VxWorks 在任务调度层次上提供了互斥访问保护的机制:在临界区内禁止任务的抢占调度,这在 VxWorks 中称为抢占上锁,具体互斥的实现是将临界区代码包括在两个函数 taskLock( )和 taskUnlock( )之间。

在 VxWorks 中,不仅任务与任务之间需要考虑互斥,同时任务与中断之间也要考虑互斥。因为中断的优先级高于任何任务,所以抢占上锁虽然保证了任务与任务间的互斥,但并没有保证任务与中断服务程序对临界区的互斥访问。VxWorks 的中断上锁则提供了最强有力的互斥,将临界区代码保护在 intLock( )和 intUnlock( )两个函数之间。中断上锁实现的是中断级互斥,在互斥期间,即使外部事件产生而引发相应的中断,系统也不会切换到相应的中断服务程序(ISR)。

因此,VxWorks 的抢占上锁和中断上锁是通过禁止任务抢占或禁止切换到中断服务程序来保护临界区的。同时,这种灵活的互斥手段在临界区中也会影响 VxWorks 任务的原始调度原则,比如低优先级任务抢占上锁时,就绪任务队列上具有较高优先级的任务却不能即时占有处理机。

2.2 Linux 的进程调度

2.2.1 Linux 进程调度的时机

Linux 进程的调度时机大致分为两种情况 :一种是进程自愿调度 ;另一种是发生强制性调度。

首先,自愿的调度随时都可以进行。在内核空间中,进程可以通过 schedule( )启动一次调度 ;在用户空间中,可以通过系统调用 pause( )达到同样的目的。如果要为自愿的暂停行为加上时间限制,在内核中使用 schedule_timeout( ),而在用户空间则使用 nanosleep( )系统调用。

Linux 中,强制性的调度发生在每次从系统调用返回的前夕,以及每次中断或异常处理返回用户空间的前夕。应注意的是,从内核态返回到用户态是进程调度发生的必要条件,而不是充分条件,还要取决于当前进程 task_struct 结构中的 need_resched 是否为 1。单 CPU 条件下,有三种情况可以使当前进程的 need_resched 置为 1[4]。

在时钟中断服务程序中,发现当前进程运行的时间片已经结束 ;当唤醒一个比当前进程优先级权值更高的进程 ;当一个进程通过系统调用改变调度策略立即引起调度时,比如系统调用 sched_setscheduler( )。

从进程调度的时机可以看出,Linux 内核的调度方式为“有条件的剥夺方式”[5]。当进程在用户空间运行,不管自愿不自愿,一旦有必要(比如时间片用完),内核就可以暂时剥夺其运行而调度其他进程运行。而进程一旦进入内核空间,即进入核心态时,尽管知道应该要调度了,但实际上却不会发生,一直要到该进程返回到用户空间前夕才能剥夺其运行。

2.2.2 Linux 进程的优先级权值

Linux 用函数 goodness( )统一计算进程(包括普通进程和实时进程)的优先级权值,该权值衡量一个处于可运行状态的进程值得运行的程度,权值越大,进程优先级越高。

每个进程的 task_struct 结构中,与 goodness( )计算权值相关的域有以下四项 :policy、nice (2.2 版内核该项为 priority)、counter、rt_priority。其中,policy 是进程的调度策略,其可用来区分实时进程和普通进程,实时进程优先于普通进程运行。nice 从最初的 UNIX 沿用而来,表示进程的静态负向优先级,其取值范围为 19~–20,以–20 优先级最高。counter 表示进程剩余的时间片计数值,由于 counter 在计算 goodness( )时起重要作用,因此,counter 也可以看作是进程的动态优先级。rt_priority 是实时进程特有的,表示实时优先级。

2.2.3 Linux 进程调度策略

首先,Linux 根据调度策略 policy 从整体上区分实时进程和普通进程。对于 policy 为 SCHED_OTHER 的普通进程,Linux 采用动态优先级调度,其优先级权值取决于(20–nice)和进程当前的剩余时间片计数 counter 之和。进程创建时,子进程继承父进程的 nice 值,而父进程的 counter 值则被分为二半,子进程和父进程各得一半。时间片计数器 counter 每次清零后由 (20–nice)经过换算重新赋值。字面上看,nice 是“优先级”、counter 是“计数器”的意思,然而实际上,它们表达的是同一个意思:nice 决定了分配给该进程的时间片计数,nice 优先级越高的进程分到的时间片越长,用户通过系统调用 nice( )或 setpriority( )改变进程静态优先级 nice 值的同时,也改变了该进程的时间片长度 ;counter 表示该进程剩余的时间片计数值,而 nice 和 counter 综合起来又决定进程可运行的优先级权值。在进程运行过程中,counter 不断减少,而 nice 保持相对不变 ;当一个普通进程的时间片用完以后,并不马上根据 nice 对 counter 进行重新赋值,只有所有处于可运行状态的普通进程的时间片都用完了以后(counter 等于 0),才根据 nice 对 counter 重新赋值,这个普通进程才有了再次被调度的机会。这说明,普通进程运行过程中,counter 的减小给了其它进程得以运行的机会,直至 counter 减为 0 时才完全放弃对 CPU 的使用,这就相当于优先级在动态变化,所以称之为动态优先调度。

对于实时进程,Linux 采用了两种调度策略,即 SCHED_FIFO(先来先服务调度)和 SCHED_RR(时间片轮转调度)。因为实时进程具有一定程度的紧迫性,所以衡量一个实时进程是否应该运行,Linux 采用了一个比较固定的标准,即参考 rt_priority 的值。Linux 用函数 goodness( )计算进程的优先级权值时,对实时进程是在 1000 的基础上加上 rt_priority 的值,而非实时进程的动态优先级综合起来的调度权值始终在 1000 以下,所以 goodness( )的优先级权值计算方法确保实时进程的调度权值

始终比所有的非实时进程都要大，这就保证了实时进程的优先运行。实时进程的 counter 与 nice 都与其优先级权值无关，这和普通进程是有区别的，实时进程 task_struct 中的 nice 和 counter 只与 SCHED_RR 调度策略进程的时间片计数相关；而对于 SCHED_FIFO 调度策略的实时进程没有调度的参考意义。

3 POSIX 调度接口

POSIX1003.1b 实时扩展标准定义了进程调度的标准函数接口形式，VxWorks 的任务调度和 Linux 中实时进程的调度都支持 POSIX1003.1b 标准，但二者在该标准的实现上差异较大，列出如下。

1）实时优先级数。VxWorks 任务的优先级为 0~255，Wind 内核默认为数值越大，任务优先级越低。Linux 实时进程优先级为 1~99，数值越大，进程优先级越高。

2）调度策略的种类。Linux 支持 SCHED_RR、SCHED_FIFO 和 SCHED_OTHER 的调度策略，而 VxWorks 只支持前两种策略，不支持 SCHED_OTHER 策略。另外 Linux 实时进程和普通进程可以互相转换，通过 SCHED_OTHER 参数调用 sched_setscheduler( )函数，可以将 Linux 实时进程转化为普通进程。通过以 SCHED_RR 或 SCHED_FIFO 为参数调用 sched_setscheduler( )也可以将普通进程转为实时进程。

3）调度策略所基于的对象。VxWorks 的任务调度策略不是基于某个任务的，而是针对整个系统的所有任务。VxWorks 下不能通过 sched_setscheduler( )来改变内核当前的任务调度策略，VxWorks 下改变调度策略的唯一手段是通过 kernelTimeSlice( )函数设置轮转调度的时间片长度来实现的，该时间片长度设为 0 时，即取消了 SCHED_RR 策略。Linux 中的调度策略是基于进程的，某个进程调度策略的设置不影响其他进程。

4 结束语

VxWorks 作为嵌入式实时操作系统，Linux 作为通用分时操作系统，二者在任务调度机制方面差异较大，包括任务上下文内容、任务的优先级、调度的时机、调度的策略等。二者在兼容 POSIX1003.1b 进程调度标准的同时其具体实现又有差异。本文通过分析与比较 Linux 和 VxWorks 任务（进程）调度机制的异同，有助于理解实时内核和通用分时内核在任务调度机制实现层次上的差异，并基于它们更有针对性地开发相应的应用，同时对基于通用操作系统 Linux 的一些实时化改造工作也提供参考借鉴作用。

参考文献

1 张尧学，史美林.计算机操作系统教程(第 2 版).清华大学出版社，2001
2 李方敏.VxWorks 高级程序设计.清华大学出版社，2004
3 Andrew S.Tanenbaum.现代操作系统.机械工业出版社，2000
4 代玲莉，欧阳劲.Linux 内核分析与实例应用.国防工业出版社，2002
5 毛德操，胡希明.Linux 内核源代码情景分析.浙江大学出版社，2001
6 Wind River System Inc.VxWorks Programmer´s Guide 5.4

[收稿日期：2005.3.7]

（上接第 29 页）

3）opc 服务器名称：填入温度巡检 OPC 服务器的注册名称 HUA.DA2.1；

4）opc 组名称：填入管理点项目的组名称（可任意起名）Group1；

5）刷新时间：1000ms；

6）死区：输入一个百分值如果过程数据值的变化不超出这个百分值将不对其进行数据更新。例如：5%；

7）本地标识：是一个默认值，不需设置；

8）同步方式和异步方式：选异步；

9）对于已经建立的 OPC 设备，可以对其配置进行修改；如果您确认不再需要，可以将它删除。修改或删除 OPC 设备的方法与其他设备的方法相同。

（2）数据连接

在导航器中双击“实时数据库/数据库组态”，然后选择“模拟 I/O 点”，双击任一空的点参数单元格，选择其“数据连接”页，出现设置对话框时进行如下操作：

1）在“连接 I/O 设备”的“设备”下拉框中选择设备 Temperature；

2）在“连接项”右侧单击“增加”按钮，出现对话框：在“OPC 点变量类型”中选择一种数据类型（可以选择“任意”以察看所有数据类型的数据项）后，双击某一数据区，数据区内指定数据类型的数据项会自动显现在右下侧的列表框中。在列表框中选择一个数据项并双击，此时系统自动生成一个完整的数据项描述并加在“数据项”输入框内；

3）在“访问路径”中输入数据的访问路径；

4）在“读写权限”中选择一种读写方式。最后单击“确定”按钮，便生成了一个数据项的数据连接；

5）重复以上步骤，为每一个需要连入热网的现场数据进行数据连接。

通过以上步骤，我们就为所有需要上网的现场数据建立了与热网实时数据库的连接。剩下的工作就是对这些实时数据的合理处理，包括画面显示、报表组态、WEB 方式发布等。

在 Web 服务器上需要安装力控软件，同时 Web 服务器保存力控发布的 HTML 文件，传送文件所需数据，并为用户提供浏览服务的站点。使用力控提供的 Web 功能，可以灵活地构建 Intranet/Internet 应用。这样应用系统的所有组态和开发工作全部在 Web Server 上进行，软件也只安装在 Web Server 上，易于系统的维护。

为了本项目将来的扩展，也为了与热网中其他的数据库进行数据共享，我们选购了力控的实时数据库转储工具——ODBCGate。ODBCGate 是力控的组件之一，它支持 Microsoft 的开放数据库互连（Open Database Connectivity–ODBC）接口，允许访问其它支持 ODBC 接口的 DBMS 系统或数据文件，它可以实现以下几个功能：历史转储、实时转储。

6 结束语

通过本项目的实施过程可以看出，OPC 技术的使用使得原来比较困难的数据集成工作变得简单明了。OPC 作为一座桥梁将企业的 MES 层与现场各种不同的厂家的控制、检测等系统有机的联系在一起，而通过 PCAuto 开发工具包的使用也使得 MES 层上的开发能够以组态的形式进行，这对于工程技术人员掌握以及管控一体化在企业中的推广是极为有效的。

参考文献

1 潘爱民.COM 原理与应用[M].北京：清华大学出版社，1999
2 三维力控编写.PCAuto3.6 编程手册
3 阳宪惠.工业数据通信与控制网络.[M]北京：清华大学出版社，2003

[收稿日期：2005.3.30]