ISSN 1009-3044 E-mail: ycjl@dnzs.net.cn

Linux 内核新旧工作队列机制的剖析和比较

马俊强

(厦门大学 信息科学与技术学院,福建 厦门 361005)

摘要:在中断驱动的程序设计中,工作队列是一种强有力的工具。但是在Linux2.6.35及其以前的内核版本中,每创建一个工作队列就创建与CPU数目相同的内核线程,耗费大量的内核资源;工作只能严格串行的处理,效率低。为了适应大规模多处理器硬件平台,提高处理效率,Linux2.6.36内核开发了受控并发工作队列机制。这种新机制由内核根据需要创建或销毁线程,工作可以并发的处理,可望替代之前长期使用的专用线程工具。文章详细介绍和剖析新工作队列机制,并通过实验,对比新旧工作队列机制的资源消耗和工作效率。结果表明,新工作队列机制大大减少内核资源的耗费,提高了处理效率。

关键词: Linux 内核; 中断驱动; 工作队列; 受控并发工作队列

中图分类号:TP316.81　文献标识码:A　文章编号:1009-3044(2014)06-1227-04

Analysis and Comparison of the New and Old Work Queue in Linux Kernel

MA Jun-qiang

(School of Information Science and Engineering, Xiamen University, Xiamen 361005, China)

Abstract: Work Queue is a powerful tool in interrupt-driven programming. But in Linux kernels of 2.6.35 and before, it consumes a lot of kernel resources in that whenever a Work Queue is created, the same number of kernel threads as CPUs are created. The efficiency to processing works in Work Queue is low due to strictly sequential processing. In order to adapt the hardware platforms with large-scale multiprocessors and increase the processing efficiency, the Linux kernel of 2.6.36 developed Concurrency Managed Workqueue. With this new mechanism, the kernel creates and destroys the threads according to the processing requirements, and works can be concurrently processed. The new Work Queue is hoped to replace the private thread tools that had been used long time. The paper introduces and analyses this new mechanism in detail, and compares the resource consumes and processing efficiency between the new and old mechanisms by experiments. The results show that the new mechanism of Work Queue greatly reduces consumes of kernel resources and increases the processing efficiency.

Key words: Linux kernel; interrupt-driven; work queue; concurrency managed workqueue

当一个中断发生时,并不是相关的各个操作都具有相同的急迫性,把所有的操作都放进中断处理程序本身也不合适。Linux内核将整个中断处理流程简单分为两个部分,第一部分是中断处理程序,称为上半部(Top Half),在运行时禁止可屏蔽中断,用于完成关键性的、紧急的处理;其余部分称为下半部(Bottom Half),在运行时开中断,主要完成与中断处理密切相关的工作,即延后执行工作(Deferring Work)。Linux内核提供三种不同形式的下半部实现机制:软中断(Softirq)、Tasklet和工作队列(Work Queue)。其中软中断和Tasklet运行在中断上下文中,不可阻塞。而工作队列由特殊的内核线程—工作者线程(Worker Thread)来执行,运行在进程上下文,可以阻塞。但是由于旧工作队列机制的缺陷,其应用并不普遍,取而代之的是使用专用线程池,如flush-x:y,bdi-default等。新工作队列机制为内核提供一种通用的线程池机制,以替代这些专用线程池[1]。实际上,只要涉及中断驱动的程序设计,工作队列都是一种强有力的工具,比如在实时控制系统中,可以创建工作队列方便高效地响应和执行由外部事件驱动的任务。文章将重点介绍和分析新工作队列机制。对新旧工作队列机制的介绍分析,选择的内核版本分别是2.6.36和2.6.35。

工作队列的设计思想可以类比于现实中的生产流水线[2]:流水线相当于工作队列中的worklist链表,加工部件相当于中断发生时所产生的工作序列,工人就是工作者线程。当中断发生时,内核将本次中断延后执行的工作序列,放到worklist工作链表中,唤醒工作者线程执行工作。工作者线程在执行时可能阻塞;当worklist中的工作处理完毕后,工作者线程进入空闲状态。从Linux 2.5.41内核引入工作队列直到2.6.35内核,其运行机制没有大的改动,主要缺点是,在有N个CPU的计算机中,每当创建一个工作队列时,就创建N条流水线,为每条流水线创建一个工作者线程,内核可以向这N条流水线提交该种类的工作(由响应中断的CPU决定)。因此,如果创建X个工作队列,则需要创建N * X个工作者线程,但是只有当流水线上有工作时,线程才运行,其余流水线上

收稿日期:2014-01-18
作者简介:马俊强(1989-),男,河南济源人,硕士研究生,主要研究方向为系统软件、分布式计算。

的线程都处于空闲状态。大量的线程需要消耗线程的ID资源和大量内存,同时也会增加调度器的负担。这种情况在拥有大量CPU的超级计算机上显得尤为浪费。另一方面,一条流水线上只有一个工作者线程,因此同一流水线上工作的处理是严格串行的,严重制约处理的效率。

为适应大规模多处理器硬件平台,提高工作队列的处理效率,从2.6.36内核开始对工作队列进行彻底的改造。在新的工作队列机制中,内核始终维持N + 1条"工作流水线",即全局每CPU工作队列gcwq(Global Percpu Workqueue,详见1.1节)。新机制的流水线是通用的,所有来自同一个CPU的中断所产生的工作序列,都放在这条流水线上。每条流水线上的工作者线程"按需分配",即当一个工作者线程阻塞时,可以让另一个工作者线程来处理该流水线上剩余的工作。当流水线上需要新的工作者线程时,就创建新线程;而当流水线上线程过多时,就销毁线程。同一条流水线上可以有多个工作同时被指派给多个工作者线程,当然任何时刻一条流水线上只有一个工作者线程在运行。这种新的工作队列机制称为受控并发工作队列(Concurrency Managed Workqueue)[3-5]。

1 受控并发工作队列

1.1 全局每CPU工作队列(gcwq)

如果计算机有N个CPU,则内核创建N + 1个gcwq,其结构如下所示:

```
struct global_cwq {
spinlock_t              lock;
unsigned int            cpu;
struct list_head        worklist;
int                     nr_workers;
int                     nr_idle;
struct list_head        idle_list;
struct hlist_head       busy_hash[BUSY_WORKER_HASH_SIZE];
struct timer_list       idle_timer;
struct timer_list       mayday_timer;
...
} ____cacheline_aligned_in_smp;
```

N个gcwq分别与N个CPU一一绑定,管理相关CPU上的工作者线程和中断产生的工作;第N + 1个gcwq称为unbound_global_cwq,其中的工作者线程未与特定的CPU绑定,详见1.4节WQ_UNBOUND标志说明。虽然gcwq也称作"工作队列",但是与用户创建的工作队列不同,它是内部管理结构,对用户不可见。gcwq中的cpu字段表示与其关联的CPU编号,worklist双向链表存储由中断提交到该CPU上的工作,lock字段为保护gcwq结构体的自旋锁。每个gcwq都维护管理一个工作者线程池,其中的工作者线程有idle(空闲)和busy(工作)两种状态;idle_list双向链表中管理处于idle状态的工作者线程,nr_idle记录其数量;为了快速检索,使用busy_hash哈希链表管理处于busy状态的工作者线程。这些线程负责处理worklist链表中工作。nr_workers记录工作者线程池中线程的数量。

1.2 新工作队列机制的运行

当中断发生时,内核调用queue_work函数将工作序列提交到gcwq。若相关的线程池中没有线程,则内核创建工作者线程;否则唤醒一个工作者线程(异常情况处理见1.4节WQ_RESCUER属性)。工作者线程调用worker_thread函数,该函数在执行中使用gcwq中的自旋锁进行保护,并完成以下动作:

1)线程从idle状态变为busy状态。

2)对所属的gcwq的线程池进行检查管理。设线程A是当前正在执行的线程,若worklist上有多个待处理的工作,则A检查线程池中是否还有处于idle状态的线程,如果存在,设选中的为线程B;否则A将创建并唤醒一个新线程B,B在创建后进入idle状态。因此在处理工作时,线程池中将保持至少一个处于idle状态的线程待命,以便迅速响应和处理worklist上的后继工作。

3)线程A从gcwq的worklist链表中依次取出未处理的工作进行处理。当处理完worklist中所有的工作后,将再一次对线程池进行检查管理,然后进入idle状态并休眠。

4)一旦线程A阻塞且worklist上还有待处理的工作,则线程B开始运行,它调用worker_thread函数重复以上过程。

如3)所述,当工作者线程处理完全部工作后将对线程池进行一次检查管理。此时如果gcwq中空闲的工作者线程过多,其判断条件是nr_idle > 2且(nr_idle - 2) * 4 >= nr_idle,则gcwq将销毁idle状态持续时间超过5分钟的工作者线程。每个gcwq的线程池最终将维护两个处于idle状态的工作者线程。

1.3 新工作队列机制的改进

1)由内核根据处理需求,控制工作者线程的创建和销毁,避免创建过多的内核线程。在工作队列空闲时,新机制中的线程数大致为(N + 1) * 2,工作队列数量与内核线程数基本无关(除非工作队列设置WQ_RESCUER标志,见1.4节)。这个改进大大减少内核资源的消耗。

2)同一CPU上一个工作队列中的工作可以并发的处理。这种并发处理方式相比旧机制的严格串行,提高了处理效率。在每个CPU上,一个工作队列中可并发处理的工作数目是有限制的(见1.4节max_active参数),当达到限制时,将不再唤醒新的工作者

线程。

3)创建工作队列时可以指定工作队列的属性。用户可以根据工作性质的不同创建不同的工作队列,如高优先级的(WQ_HIGHPRI)、未绑定的(WQ_UNBOUND)、不可重入的(WQ_NON_REENTRANT)、带救援者线程的(WQ_RESCUER)[3][5]等。在图1中,gcwq的worklist链表中的工作分为高优先级的工作和普通优先级的工作两类,高优先级的工作排在链表头部,普通优先级的工作排在链表尾部,同一类别的工作之间按照提交的顺序排列。而在旧工作队列机制中则没有这些属性,工作按照提交的顺序被执行。

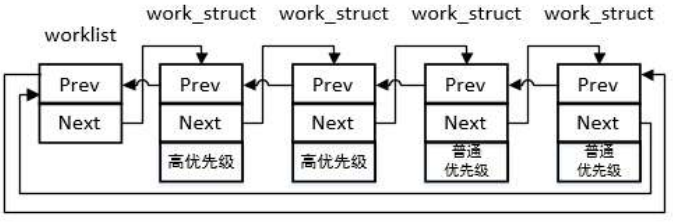

图1 gcwq的worklist链表中的工作分布

4)新工作队列机制提供4个预定义工作队列,方便用户使用工作队列。这四个预定义的工作队列分别为events、events_long、events_nrt、events_unbound。其中events是普通工作队列,要求其中的工作执行时间尽量短;需要长时间处理的工作可以提交到events_long队列中;events_nrt是不可重入工作队列,其中的工作将不会在多个CPU上并发执行;events_unbound队列就是设置了WQ_UNBOUND标志的队列,只要处理的工作数量未达到限制,其中的工作就可以立即被处理。用户可以根据需要使用这4个预定义工作队列,当然也可以自己创建工作队列。

1.4 创建工作队列

调用下面的宏创建一个工作队列:alloc_workqueue(name, flags, max_active)。

参数name:指定工作队列的名称。

参数flags:标志位,指明工作队列的属性,摘要解释如下:

WQ_NON_REENTRANT:默认时,工作队列中的多个同一种类的工作,在一个CPU上不可重入,但是允许在不同CPU上重入(即允许在不同CPU上并发的处理)。如果设置此标志,则工作队列中的多个同一种类的工作在不同CPU上也不可重入。此时工作可能需要从响应中断的CPU迁移到另一个CPU上:当中断产生工作时,如果以前产生的同类工作正在另一个CPU上处理,则将该工作提交到这个CPU上。

WQ_UNBOUND:如果设置此标志,则内核将为工作队列设置WQ_HIGHPRI标志,其上的工作都将插入到unbound_global_cwq的worklist链表上。unbound_global_cwq中工作将按照提交的顺序被处理。通常,为了更好地利用CPU缓存,工作在所提交的CPU上处理,而unbound_global_cwq中的工作者线程未与特定的CPU绑定,其中的工作可能运行在任意一个CPU上,因此无法有效利用CPU缓存。此标志是为需要大量CPU周期的工作设置的,此时各个CPU的负载均衡更为重要,所以此类工作最好由调度器决定在哪一个CPU上运行。

WQ_RESCUER:如果设置此标志,则为工作队列专门创建一个救援者线程(Rescure Thread)。创建救援者线程的目的是避免长时间的等待或死锁。由于内核创建工作者线程时使用GFP_KERNEL标志来分配内存,可能导致创建过程长时间阻塞,因此在创建时,内核设置一个定时器mayday_timer。如果定时器超时,但线程仍未创建成功,那么内核就唤醒各个工作队列中的救援者线程,执行rescuer_thread函数处理的剩余工作。所有在处理时可能与内存回收执行路径重叠的工作队列,必须设置这个标志。

WQ_HIGHPRI:如果设置此标志,则该工作队列中的工作都是高优先级的,其中的工作将被插入到目标gcwq的工作列表worklist的最后一个高优先级工作后面,即高优先级的工作排在worklist队列头,且依据提交的顺序被处理。只要资源可用,总是尽可能快地处理高优先级工作。

参数max_active:一个工作队列在每个CPU上可能并发执行的最大工作数目。对绑定的工作队列,max_active最大值512;默认值为0,此时max_active为256;对于未绑定的工作队列,max_active的最大值为max(512, 4 * num_possible_cpus())。建议内核开发者使用默认值。

2 实验设计和结果

通过两个实验来对比新旧工作队列机制的资源消耗和工作效率,实验环境如表1所示。

表1 实验环境

	AMD Athlon 64	I5-3470 3.20GHz	I7-860 2.8GHz
CPU数	2	4	8
系统	Ubuntu	Ubuntu	Ubuntu
内存	2 G	4 G	16 G

2.1 新旧工作队列机制中可创建的最大工作队列数

表2 最大工作队列数

	旧工作队列机制			新工作队列机制		
CPU数	2	4	8	2	4	8
最大工作队列数	14874	8041	4048	79838	112605	3775985
创建线程数	29748	32164	32384	0	0	0

在本实验中,测试机器的pid_max设置为32768。由表2可以看出,在旧工作队列机制中,可创建的工作队列数目随着CPU数目的增加而减少,主要受限于内核可创建的线程数;在新工作队列机制中,可创建的工作队列数目随着内存的增加而增加,与系统可创建的最大线程数无关。新工作队列机制中的最大工作队列数主要受限于内核可分配的percpu空间。这是因为每创建一个工作队列都需要分配一定大小且地址对齐的percpu空间,随着创建的工作队列数的增加,内核中将没有可用的percpu空间,从而导致创建工作队列失败。

2.2 在新旧工作队列机制中,同一CPU上一个工作队列中10个工作的执行效率的比较

实验所用的工作队列不带任何标志位,每个工作分别有三种休眠时间0s、1s或5s,用来模拟处理的时间。新工作队列机制的并发工作数有1、5、256三种情况。实验结果见表3。

表3 同一CPU上一个工作队列中10个工作的处理时间

	10个工作完成所需时间(s)			
工作睡眠时间(s)	旧工作队列机制	新工作队列机制(max_active)		
		1	5	256
0	0.000132	0.000133	0.000148	0.000131
1	10.009443	10.010228	2.005546	1.003993
5	50.009821	50.009263	10.005712	5.002429

如果所有工作都不休眠或者max_active等于1,则新旧工作队列机制下10个工作的处理时间基本相同。当工作有休眠且max_active大于1时,一旦处理工作的线程休眠,则内核立即唤醒新的线程执行后继的工作,直到正在执行的工作数等于max_active。实验结果表明,新工作队列机制可以显著提高处理的效率。

3 结束语

旧工作队列机制应用于大规模多处理器硬件平台会耗费大量的内核资源,工作的处理效率也很低。以往的补救方法是使用专用线程工具。Linux2.6.36内核开发受控并发工作队列,由内核"按需分配"工作者线程,大大减少内核资源的耗费;同时,工作可以并发处理,提高了处理效率。新机制提供的通用的线程池有望替代内核中专用线程池,成为中断驱动程序设计的强有力的工具。

新工作队列机制的设计也存在一些缺陷。首先,优先级只分为高优先级和普通优先级,略显粗糙,无法区分一个工作队列中不同工作之间的差别。其次,工作队列的优先级与内核线程的优先级无关,在资源比较紧张时,可能无法满足硬实时任务的需要。再有,max_active参数的设置不是针对一个工作队列,而是针对一个CPU上所有工作队列,也不够灵活。如果这些问题得到改进,新机制将能更方便地应用于实时系统。

参考文献:

[1] Tejun Heo .backing-dev: replace private thread pool with workqueue.txt [EB/OL]. [2010-09].http://lwn.net/Articles/403653.

[2] 陈学松.深入Linux设备驱动程序内核机制[M].北京:电子工业出版社,2012:214-230.

[3] Tejun Heo.workqueue.txt [EB/OL]. [2010-09].http://lxr.linux.no/linux+v-2.6.36.4/Documentation/workqueue.txt.

[4] Jonathan Corbet .Concurrency-managed workqueues [EB/OL]. [2010-09].http://lwn.net/Articles/355700.

[5] Jonathan Corbet .Working on workqueues [EB/OL]. [2010-09].http://lwn.net/Articles/403891.