计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(11):115−119 [doi: 10.15888/j.cnki.csa.006606]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

对 EAST PCS 系统延迟的分析与改善①

唐剑寅[1,2], 肖炳甲[1,2], 袁旗平[1]

[1](中国科学院 合肥物质科学研究院 等离子体物理研究所, 合肥 230031)
[2](中国科学技术大学, 合肥 230026)

摘　要: 在等离子体控制系统 PCS 中, 操作系统噪声通常带来几微秒到几十微秒延迟, 无法满足日益严格的控制要求. 基于 Linux 内核机制, 可推断延迟的产生主要由于调度和中断. 为避免调度, 对控制进程采用实时调度, 并进行配套系统调整; 为减少中断, 进行中断迁移, 并使用修改了内核滴答频率的 Linux 内核. 最终延迟情况得到有效改善, 从而为数据采集、计算和传输保留了充裕时间.

关键词: 等离子体控制; 操作系统噪声; 延迟; 调度; 中断

引用格式: 唐剑寅,肖炳甲,袁旗平.对 EAST PCS 系统延迟的分析与改善.计算机系统应用,2018,27(11):115–119. http://www.c-s-a.org.cn/1003-3254/6606.html

Analysis and Reduction of OS Latency in EAST PCS

TANG Jian-Yin[1,2], XIAO Bing-Jia[1,2], YUAN Qi-Ping[1]

[1](Institute of Plasma Physics, Hefei Institutes of Physical Science, Chinese Academy of Sciences, Hefei 230031, China)
[2](University of Science and Technology of China, Hefei 230026, China)

Abstract: Inside the Plasma Control System (PCS), Operating System (OS) noises usually bring latency measured in microseconds, which make the PCS hard to satisfy the increasing complexity of advanced plasma control requirements. Analysis shows that schedule and interrupts are the main cause of latency. In order to avoid scheduling, real-time priorities are applied in control processes, and related system adjustments are done. In order to decrease the frequency of interrupts, device interrupts are migrated and a revised kernel with fewer ticks is adopted. These work turn out to be efficient to reduce latency, and sufficient time is reserved for data acquisition, computation, and transmission.

Key words: Plasma Control System (PCS); Operating System (OS) noise; latency; scheduling; interrupt

1　引言

EAST (Experimental Advanced Superconductive Tokamak) 是我国自行设计研制的全超导托卡马克装置, 目前进行周期 100 μs, 总时长最长 1000 s 的放电实验. 其等离子体控制系统 PCS (Plasma Control System) 负责从外部子系统获取数据并运行控制算法, 最终将结果反馈到外部系统.

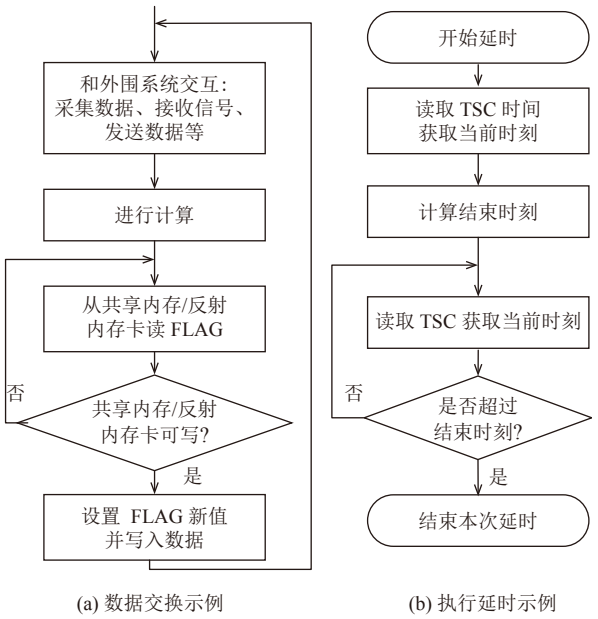
1.1　EAST PCS 软硬件特性

EAST PCS 在硬件平台上, 使用 x86 Linux 服务器作为控制进程执行环境, 外部设备则使用不依赖中断工作的轮询设备, 包括采集卡和反射内存卡. 系统方面使用定制的 2.6 内核 Linux, 该系统主要通过模块方式增加了一个屏蔽外部设备中断的接口, 供控制进程调用[1].

较短的放电周期给系统实时性提出一定要求, PCS 中控制进程选择采用循环轮询 (Polled Loop) 方式运行[2], 相较于中断事件驱动型程序, 这种方式采用循环等待的方式获取资源, 持续持有 CPU, 而非频繁进入

① 基金项目: 国家重点基础研究发展计划项目 (973 计划)(2014GB103000)
Foundation item: National Program on Key Basic Research Project of China (973 Program) (2014GB103000)
收稿时间: 2018-03-24; 修改时间: 2018-04-24; 采用时间: 2018-04-27; csa 在线出版时间: 2018-10-24

中断, 在 PCS 这类数据量大、计算任务重的快速系统中能获得更稳定的实时能力. 在放电执行阶段, 根据算法不同, 多个相应控制进程进入实时控制状态并行运行, 简单示例如图 1, 控制进程在获取数据时, 循环读取共享内存或反射内存卡以等待数据到来; 进行延时操作时, 循环读取 TSC (Time Stamp Counter) 得到当前时间, 直到超时.



(a) 数据交换示例 (b) 执行延时示例

图 1 控制进程工作方式示例

控制进程还进行了 CPU 亲和性设置, 总是运行在特定处理器核上不发生迁移. 并且采用了 mlockall 的方式锁定内存, 避免在运行过程中发生内存缺页, 进行处理时延误控制进程运行.

1.2 PCS 的延迟问题

循环轮询进程期望获得所有处理器时间, 但 Linux 内核无法保证某个任务不受干扰地持续运行, 系统噪声成为了 PCS 控制进程的瓶颈. Linux 系统噪声的产生是由于系统本身为了维持正常运行, 偶尔需要占用处理器一段时间, 在进程中就表现为延迟[3]. 从系统层面而言, 控制进程产生延迟的直接原因, 无非是其它进程或中断占用处理器[4]. 在当前 100 μs 控制周期下, 这些延迟压缩了控制进程数据采集、计算和传输的可用时间, 甚至导致控制进程周期内无法完成工作, 影响控制效果. 而 PCS 还计划采用 50 μs 周期控制周期, 这一需求又将进一步放大噪声影响.

1.3 国内外现状

EAST PCS 和韩国 KSTAR PCS 都源自美国的 DIII-D PCS, 三者有着类似的软硬件特性.

DIII-D PCS 目前采用 CentOS 6 系统, 内核为 2.6.32. 为减少系统噪声, DIII-D PCS 使用了 isolcpus 内核启动参数以避免系统自动分配任务至控制进程所在处理器, 并设置了设备中断亲和性, 以避免设备中断干扰.

KSTAR PCS 目前采用 Scientific Linux 6 系统和 CERN MRG 实时内核, 内核版本为 3.10.33, 其采用实时内核主要因为使用了产生中断的设备[5]. KSTAR PCS 中的控制进程采用了实时调度以避免控制进程被意外调度, 并且同样使用了内核启动参数 isolcpus.

二者对于 EAST PCS 进行系统噪声改善有一定借鉴意义, 尤其是实时优先级的加入和设备中断迁移的做法值得应用. 然而 EAST 的控制周期更短, 实验时间更长, 并且三个 PCS 的控制进程也互有区别. EAST PCS 若武断提升控制进程为实时优先级容易导致进程阻塞等问题, 并且仅仅禁止调度可能仍无法满足控制需求.

为控制进程添加实时优先级并设法减少中断发生是减少系统噪声的基本思路, 本文通过在 PCS 代码中选择恰当时机设置实时优先级, 并进行相应系统调整, 避免可能出现进程或系统阻塞问题, 另外本文还通过迁移设备中断、使用无滴答内核让中断发生情况得到大幅改观, 最终通过对比测试, 证明了本文方法切实有效.

2 延迟改善分析

2.1 改善进程影响

EAST PCS 控制进程未特别指定调度方式, 其所在核上运行队列中的进程可能带来影响. 运行队列中的进程一般来自两种情况:

(1) 内核为了保证系统负载均衡, 会将进程分配到各个处理器核上;

(2) 内核在各个核会固定维持一些守护进程, 并且不时唤醒.

由于 Linux 中的完全公平调度器 CFS (Completely Fair Scheduler) 趋向于为运行队列中进程公平分配 CPU 时间[6], 这两种进程都可能导致控制进程产生被动上下文切换 (involuntary context switch), 从而存在延迟.

使用实时调度方式能减少控制进程受到的无关进程影响, Linux 内核一般至少提供 SCHED_FIFO 和

SCHED_RR 两种实时调度方式. 采用这两种调度方式的任务能比普通任务优先执行, 并且高优先级实时任务总是能够抢占低优先级实时任务. 对于同一优先级的任务, 采用 SCHED_RR 方式的任务会和其它任务轮流运行, 而采用 SCHED_FIFO 的任务采用先进先出的方式运行, 直到任务完全结束或者主动放弃 CPU, 才会允许下一个任务投入运行[7].

2.2 改善中断影响

在 Linux 系统中, 中断永远比进程优先被处理, 因此实时优先级无法避免中断带来的延迟.

由于外部设备中断被屏蔽, 目前 EAST PCS 中发生最为频繁的是本地时钟中断 LOC (Local timer interrupt). 系统利用 LOC 中断进行状态更新、信息统计、内存回收、任务调度等工作[8]. LOC 中断的频率取决于内核编译时 CONFIG_HZ 的值, 在 EAST PCS Linux 系统中 LOC 中断频率默认为 1000 HZ.

自 3.10 内核以后, Linux 支持完全动态滴答特性 (Full Dynamic ticks), 即当前核上的运行队列中不超过 1 个任务时, 将 CPU 的 LOC 中断设置为 1 Hz[9], 该特性大大降低了中断频率. 在一些看重延迟或吞吐量的高性能计算场景中, 该特性一般会被应用, 以减少系统噪声, 获得更稳定的性能[10].

完全动态滴答特性开启后, 仍会保持 1 Hz 的时钟中断滴答, 用于更新 vruntime、负载信息等, 低精度定时任务也依赖该滴答. 由于 EAST PCS 专用于控制, 对调度的和负载信息统计的准确性及时性并不要求, 因此可将 1 Hz 的时钟滴答完全消除以进一步减少控制进程抖动.

3 实时调度加入

PCS 采用 SCHED_FIFO 调度和最高实时优先级调度控制进程, 以完全避免控制过程中进程切换.

PCS 代码中添加实时调度的时机有所选择, 主要因为内核对 mlockall 的实现, 与 PCS 控制进程采用的循环轮询方式存在冲突. Linux 内核自 2.6.28 版本以后, mlockall 的实现中调用了 lru_add_drain_all 函数, 该函数负责将 pagevec 中的页转移到相应 lru 链表中, 它将这一工作交给每个核上的内核工作队列线程, 并等待其执行结束. 然而内核工作队列线程并非实时优先级, 会被已运行的实时循环轮询进程阻塞, 从而导致 mlockall 无法执行结束. 因此在 PCS 代码中需保证所有控制进程的 mlockall 执行结束, 才能提升任务为实时优先级. 在 PCS 中, 控制进程的工作分为三部分, 首先是设置阶段, 这一阶段控制进程预先申请所需内存和其它资源, 并且使用 mlockall 锁定整个内存; 第二阶段是执行阶段, 进行等离子体控制; 第三阶段是清除阶段, 进行资源释放和数据存储[11]. 设置阶段所有控制进程都会完成初始化, mlockall 也在初始化阶段调用, 完成设置的进程进入轮询状态等待执行阶段开始. 若控制进程直接以实时优先级运行, 最快完成内存锁定的进程会以实时优先级持续持有 CPU, 其它进程的内存锁定工作便因此阻塞. 因此实时优先级的设置应当在初始化阶段之后, 进入执行状态之前, 才能保障进程不发生阻塞.

EAST PCS 控制进程采用实时优先级后, 成为实时循环轮询进程, 并在运行时希望获得全部的 CPU 时间. 而内核的默认配置并不保证这点, 内核对实时进程的运行时间进行了限制, 让每秒中有部分时间用于普通进程. 以下命令可以解除该限制[12]: echo-1 > /proc/sys/kernel/sched_rt_runtime_us

控制进程提升至实时优先级后, 相应核上其它进程得不到运行, 这可能导致系统无法正常工作. 在 3.10 内核中, 还需进行如下配置:

(1) 避免系统自动分配进程到相应核上. 在内核启动参数中添加"isolcpus=1–*N*", *N* 的值为处理器核数减一. 这里仅保留第 0 个处理器核为守护核, 接管非控制任务.

(2) 避免系统唤醒控制进程核上的守护进程. 系统会唤醒守护进程执行日常事务活动[13], 一些工作具有 NUMA 亲和性, 可能导致工作无意识地分配在非守护核上而被阻塞. 在内核启动参数中添加"numa=off", 以避免该情况. I/O 可能在不同核上唤醒处理, 通过以下命令进行设置, 避免 I/O 队列放到非守护核: echo 1 > /sys/bus/workqueue/devices/writeback/cpumask.

(3) 关闭系统检测机制. 由于放电实验最长可至 1000 s, 控制进程在此期间不放弃处理器, 如此长的时间会让 Linux 系统中 watchdog 判断系统挂起. 使用如下命令关闭 watchdog:

```
echo 0 > /proc/sys/kernel/watchdog
echo 0 > /proc/sys/kernel/nmi_watchdog
```

4 中断频率降低

PCS 升级内核至 3.10 版本以获得完全动态滴答特性, 并更改了对外部设备中断的处理方式.

外部设备中断在 Linux 中/proc 文件系统中分配了一个数字中断号, 可以通过/proc 接口设置中断对 CPU 的亲和性, 从而实现中断迁移. 更新内核后, PCS 选择将外部设备中断迁移到守护核, 而非直接屏蔽设备中断, 因此在放电实验执行阶段, 控制进程所在服务器仍具备通过网络远程管理的能力.

为了进一步提供不受中断影响的环境, PCS 中基于 Linux 3.10 内核已有的完全动态滴答特性, 为内核添加了新的启动参数 pcs_nohz. 设置该参数后, LOC 中断频率从 1 Hz 变成 0 Hz, 从而 LOC 中断次数不再随放电实验时间增长而增加.

以 3.10 内核源代码为例, 具体修改方式如下:

(1) 进入内核源码目录, 打开 kernel/sched/core.c, 在 scheduler_tick_max_deferment 函数之前加入如下行:

```
/*标志位,指示是否开启 0 HZ 时钟中断,默认为 false*/static bool pcs_nohz;
/*内核初始化时,将根据是否设置对应启动参数选择性调用该函数*/
static __init int set_pcsnohz(char *args)
{ /*设置标志位*/ pcs_nohz = true; return 1; }
/*关联启动参数与相关函数*/ __setup
(“pcs_nohz”, set_pcsnohz);
```

(2) 在 scheduler_tick_max_deferment 函数中添加两行新代码, 修改后的函数如下:

```
/*该函数是实现完全动态滴答特性的关键函数, 它返回下次滴答的时间*/
u64 scheduler_tick_max_deferment(void)
{ struct rq *rq = this_rq(); unsigned long next,
now = ACCESS_ONCE(jiffies);
  /* 以下两行代码为新添加行,如果标志位已设置,则返回一个最大时间,表示永远不产生滴答*/
if (pcs_nohz)
return KTIME_MAX;
next = rq->last_sched_tick + HZ; if (time_before_eq
(next, now)) return 0; return jiffies_to_usecs
(next-now)* NSEC_PER_USEC;}
```

然后确保配置 CONFIG_NO_HZ_FULL=y 并编译安装内核, 在内核启动参数中添加”nohz_full=1-N pcsnohz”并重启, 即可消除时钟滴答.

5 延迟测试

PCS 系统中直接使用汇编指令 rdtsc 读取 TSC 来计时, 使用该指令读取时间至局部变量中仅耗费数十个指令周期, 而系统的噪声导致延迟的间隔一般远大于此开销. 因此可以使用连续的 rdtsc 指令得到延迟数据, 并从次数和大小两个维度进行统计.

测试平台配置如表 1 所示.

表 1 测试平台配置

服务器型号	Dell R730
处理器	双路 Intel Xeon CPU E5-2637 v3 @3.50 GHZ
内存	128 GB

测试运行时间选择 1000 s, 这是目前等离子体放电的最长时间, 同时, 有的中断周期或者系统检测机制周期可能长达数百秒, 1000 s 一般足够涵盖所有可能的系统噪声类型.

测试的内核有三个版本, 分别为:

1) 2.6.32, 维持 PCS 原来的系统配置和调度方式(屏蔽外部设备中断,使用 CFS 调度), 控制进程的原本运行环境;

2) 3.10, 屏蔽外部设备中断, 使用 CFS 调度, 模拟 PCS 直接升级内核的表现;

3) 3.10, 采用实时调度, 使用改善过中断的内核, 优化延迟后的控制进程运行环境.

PCS 中可以忽略 1 μs 以下的延迟, 在测试平台上, 使用连续读取 TSC 的方式, 测试 1000 s,仅统计 1 μs 及以上延迟, 得到结果如图 2.

从图 2 中可以发现, 2.6.32 内核大部分延迟分布在[1, 2) μs 区间, 随着延迟增大, 延迟发生次数逐步减少, 但超过 5 μs 的延迟仍有相当一部分. 使用 3.10 内核比 2.6.32 内核单次延迟时间出现增长, 这是由于 Linux 在各个版本中, 内核代码变动较多, 导致系统噪声带来的延迟并不一致, 并非越新的内核就能带来更佳效果. 而按照本文的延迟优化方法, 在 3.10 内核上, 只发生 3 次延迟, 并且所有延迟都在 5 μs 以内, 改善十分显著.

不进行延迟优化时, 表 2 中≥1 μs 总延迟次数在使用 2.6.32 和 3.10 内核时是类似的, 结合 1000 Hz 的 LOC 中断频率, 可以发现总延迟次数刚好和 LOC 中断次数接近, 这也凸显 LOC 中断对系统整体延迟的影响. 不应用优化策略情况下, 延迟最大值已经突破了 100 μs 控制周期的限制, 对控制效果必然产生影响.

经过优化, 控制进程的实时优先级保证其在执行阶段不会发生被动上下文切换, 并且无滴答的内核不再有每秒固定次数的 LOC 中断打扰控制进程, 这是测

试结果中优化前后出现差异的主要原因. 优化前后结果对比显示本文方法能够大幅减少延迟频率, 并且改善最坏延迟情况. PCS 控制进程并非每周期中所有时间都在进行运算和数据传输, 而是一部分时间用于标志位轮询, 这部分时间是弹性的. 这实际增加了 PCS 对于系统噪声的容忍度, 只有系统噪声过大或者发生在周期末尾才会导致周期超出. 而优化后即使最坏情况发生, 单个 PCS 控制进程仅仅超出周期 3 μs, 几乎不对等离子体控制产生影响.

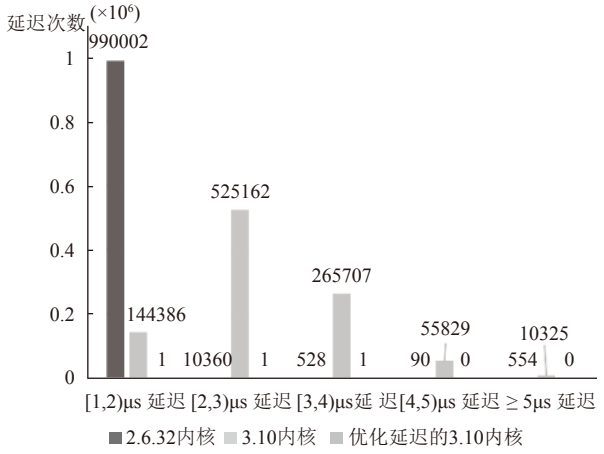


图 2 测试 1000 s 延迟分布情况

表 2 测试 1000 s 延迟总体情况

内核版本	2.6.32	3.10	优化延迟的 3.10
≥1 μs 总延迟次数	1001 534	1001 409	3
延迟最大值 (μs)	154.3	129.3	3.1
延迟平均值 (μs)	1.5	2.8	2.3

6 结语

系统噪声通常直接来自调度和中断, 会给 PCS 控制进程的执行带来延迟, 本文按照增加实时调度, 降低中断频率的基本思路来改善控制进程延迟.

加入实时调度方式需要协调好实时优先级与已有代码的关系, 合理安排控制进程提升为实时任务的时机. 同时还需调整系统, 使之与长时间实时循环轮询任务不冲突.

中断的减少有多种优化方式, 对外部设备中断这类能够直接迁移的中断只需进行一些系统配置, 而对 LOC 中断则需通过修改内核代码降低其发生频率.

优化后的测试结果显示, 进程能感知到的最大延迟下降到 3 μs 左右, 同时延迟次数极低. 经过分析, 该结果已完全符合等离子体控制需求.

在循环轮询进程的实时性方面, 本文已做到基本消除系统噪声, 但未来等离子体控制必然使用更高速的外部设备, 其中一些设备可能依赖中断工作. 那时使用一款实时操作系统, 并进一步改善中断响应和进程唤醒的实时性, 又将成为新的挑战.

参考文献

1 Penaflor BG, Ferron JR, Piglowski DA, *et al*. Real-time data acquisition and feedback control using Linux Intel computers. Fusion Engineering and Design, 2006, 81(15–17): 1923–1926. [doi: 10.1016/j.fusengdes.2006.04.008]

2 Laplante PA, Ovaska SJ. Real-time Systems Design and Analysis: Tools for the Practitioner. 4th ed. New Jersey, USA: John Wiley & Sons, 2011. 82–83.

3 Lameter C. Shoot first and stop the OS noise. Ottawa Linux Symposium. Ottawa. 2009.

4 Tsafrir D, Etsion Y, Feitelson DG, *et al*. System noise, OS clock ticks, and fine-grained parallel applications. Proceedings of the 19th annual International Conference on Supercomputing. Cambridge, MA, USA. 2005. 303–312.

5 Hahn SH, Penaflor BG, Milne PG, *et al*. Achievements and lessons learned from the operation of KSTAR plasma control system upgrade. Fusion Engineering and Design, 2018, 130: 16–20. [doi: 10.1016/j.fusengdes.2018.02.066]

6 刘婷, 王华军, 王光辉. 基于 Linux 内核的 CFS 调度算法研究. 电脑与电信, 2010, (4): 61–63. [doi: 10.3969/j.issn.1008-6609.2010.04.034]

7 Bovet DP, Cesati M. 深入理解 Linux 内核. 陈莉君, 张琼声, 张宏伟, 译. 北京: 中国电力出版社, 2007. 261–262.

8 Morari A, Gioiosa R, Wisniewski RW, *et al*. A quantitative analysis of OS noise. Proceedings of 2011 IEEE International Parallel &Distributed Processing Symposium. Anchorage, AK, USA. 2011. 852–863.

9 Corbet J. (Nearly) full tickless operation in 3.10. https://lwn.net/Articles/549580. [2015-05-08].

10 Horikoshi M, Meadows L, Elken T, *et al*. Scaling collectives on large clusters using Intel(R) architecture processors and fabric. Proceedings of Workshops of HPC Asia. Chiyoda, Tokyo. 2018. 59–62.

11 袁旗平. 基于 Linux 集群架构的等离子体控制系统[博士学位论文]. 合肥: 中国科学院合肥物质科学研究院, 2009.

12 Fayyad-Kazan H, Perneel L, Timmerman M. Linux PREEMPT-RT v2.6.33 versus v3.6.6: Better or worse for real-time applications. ACM Sigbed Review, 2014, 11(1): 26–31. [doi: 10.1145/2597457]

13 Stevens WS, Rago SA. UNIX 环境高级编程. 戚正伟, 张亚英, 尤晋元, 译. 3 版. 北京: 人民邮电出版社, 2014. 372–374.