Realtime Audio vs. Linux 2.6

Lee Revell
Mindpipe Audio
305 S. 11th St. 2R
Philadelphia, PA, 19107
USA,
rlrevell@joe-job.com

Abstract

From the beginning of its development kernel 2.6 promised latency as low as a patched 2.4 kernel. These claims proved to be premature when testing of the 2.6.7 kernel showed it was much worse than 2.4. I present here a review of the most significant latency problems discovered and solved by the kernel developers with the input of the Linux audio community between the beginning of this informal collaboration in July 2004 around kernel 2.6.7 through the most recent development release, 2.6.16-rc5. Most of these solutions went into the mainline kernel directly or via the -mm, voluntary-preempt, realtime-preempt, and -rt patch sets maintained by Ingo Molnar (Molnar, 2004) and many others.

Keywords

Latency, Preemption, Kernel, 2.6, Realtime

1 Introduction

In mid-2004 Paul Davis, and other Linux audio developers found that the 2.6 kernel, despite promises of low latency without custom patches, was essentially unusable as an audio platform due to large gaps in scheduling latency. They responded with a letter to the kernel developers which ignited intense interest among the kernel developers (Molnar, 2004) in solving this problem. Massive progress was made, and recent 2.6 releases like 2.6.14 provide latency as good or better than the proprietary alternatives. This is a review of some of the problems encountered and how they were solved. . . .

2 Background

The main requirements for realtime audio on a general purpose PC operating system are application support, driver support, and low scheduling latency. Linux audio began in earnest around 2000 when these three requirements were met by (respectively) JACK, ALSA, and the low latency patches for Linux 2.4 (”2.4+ll”). The 2.6 kernel promised low scheduling latency (and therefore good audio performance) without custom patches, as kernel preemption was available by default. However early 2.6 kernels (2.6.0 through approximately 2.6.7) were tested by the Linux audio development community and found to be a significant regression from 2.4+ll. These concerns were communicated privately to kernel developer Ingo Molnar and 2.6 kernel maintainer Andrew Morton; Molnar and Arjan van de Ven responded in July 2004 with the ”Voluntary Kernel Preemption patch” (Molnar, 2004). The name is actually misleading - ’Voluntary’ only refers to the feature of turning might_sleep() debugging checks into scheduling points if preemption is disabled. The interesting features for realtime audio users, who will always enable preemption, are the additional rescheduling points with lock breaks that Molnar and van de Ven added wherever they found a latency over 1ms.

3 Latency debugging mechanisms

The first requirement to beat Linux 2.6 into shape as an audio platform was to develop a mechanism to determine the source of an xrun. Although kernel 2.6 claims to be fully preemptible, there are many situations that prevent preemption, such as holding a spinlock, the BKL, or explicitly calling preempt_disable(), or any code that executes in hard or soft interrupt context (regardless of any locks held).

The first method used was ALSA’s ”xrun debug” feature, about the crudest imaginable latency debugging tool, by which ALSA simply calls dump_stack() when an xrun is detected, in the hope that some clue to the kernel code path responsible remains on the stack. This crude mechanism found many bugs, but an improved method was quickly developed.

In the early days of the voluntary preemption patch, Molnar developed a latency tracing mechanism. This causes the kernel to trace every function call, along with any operation that affects the "preempt count". The preempt count is how the kernel knows whether preemption is allowed - it is incremented or decremented according to the rules above (taking spinlock or BKL increments it, releasing decrements, etc) and preemption is only allowed when the count is zero. The kernel tracks the maximum latency (amount of time the preempt count is nonzero) and if it exceeds the previous value, saves the entire call stack from the time the preempt count became positive to when it became negative to /proc/latency_trace).

So rather than having to guess which kernel code path caused an xrun we receive an exact record of the code path. This mechanism has persisted more or less unchanged from the beginning of the voluntary preemption patches (Molnar, 2004) to the present, and within a week of being ported to the mainline kernel had identified at least one latency regression (from 2.6.14 to 2.6.15, in the VM), and has been used by the author to find another (in free_swap_cache()) in the past week. Dozens of latency problems have been fixed with Molnar's tracer (everything in this paper, unless otherwise noted); it is the one of the most successful kernel debugging tools ever.

4 The BKL: ReiserFS 3

One of the very first issues found was that ReiserFS 3.x was not a good choice for low latency systems. Exactly why was never really established, as the filesystem was in maintenance mode, so any problems were unlikely to be fixed. One possibility is that reiser3's extensive use of the BKL (big kernel lock - a coarse grained lock which dates from the first SMP implementations of Linux, where it was used to provide quick and dirty locking for code with UP assumptions which otherwise would have to be rewritten for SMP). ReiserFS 3.x uses the BKL for all write locking. The BKL at the time disabled preemption, which is no longer the case, so the suitability of ReiserFS 3.x for low latency audio systems may be worth revisiting. Hans Reiser claims that ReiserFS 4.x solves these problems.

5 The BKL: Virtual console switching

One of the oldest known latency issues involved virtual console (VC) switching (as with Alt-Fn), as like ReiserFS 3.x this process relies on the BKL for locking which must be held for the duration of the console switch to prevent display corruption. This problem which had been known since the 2.4 low latency patches was also resolved with the introduction of the preemptible BKL.

6 Hardirq context

Another issue discovered in the very early testing of the voluntary preemption patches was excessive latency caused by large IO requests by the ATA driver. It had previously been known that with IDE IO completions being handled in hard IRQ context and a maximum request size of 32MB (depending on whether LBA48 is in effect which in turn depends on the size of the drive), scheduling latencies of many milliseconds occurred when processing IO in IRQ context.

This was fixed by adding the sysfs tunables:
/sys/block/hd*/queue/max_sectors_kb
which can be used to limit the amount of IO processed in a single disk interrupt, eliminating excessive scheduling latencies at a small price in disk throughput.

Another quite humorous hardirq latency bug occurred when toggling Caps, Scroll, or Num Lock - the PS/2 keyboard driver actually spun in the interrupt handler polling for LED status (!). Needless to say this was quickly and quietly fixed.

7 Process context - VFS and VM issues

Several issues were found in the VFS and VM subsystems of the kernel, which are invoked quite frequently in process context, such as when files are deleted or a process exits. These often involve operations on large data structures that can run for long enough to cause audio dropouts and were most easily triggered by heavy disk benchmarks (bonnie, iozone, tiobench, dbench).

One typical VFS latency issue involved shrinking the kernel's directory cache when a directory with thousands of files was deleted; a typical VM latency problem would cause audio dropouts at process exit when the kernel unmapped all of that processes virtual memory areas with preemption disabled. The sync() syscall also caused xruns if large amounts of dirty data was flushed.

One significant process-context latency bug was discovered quite accidentally, when the author was developing an ALSA driver that required running separate JACK instances for playback and capture. A large xrun would be induced in the running JACK process when another was started. The problem was identified as mlockall() calling into make_pages_present() which in turn called get_user_pages() causing the entire address space to be faulted in with preemption disabled.

Process-context latency problems were fortunately the easiest to solve, by the addition of a reschedule with lock break within the problematic loop.

8 Process context - ext3fs

While ReiserFS 3.x did not get any latency fixes as it was in maintenance mode, EXT3FS did require several changes to achieve acceptable scheduling latencies. At least three latency problems in the EXT3 journalling code (a mechanism for preserving file system integrity in the event of power loss without lengthy file system checks at reboot) and one in the reservation code (a mechanism by which the filesystem speeds allocation by preallocating space in anticipation that a file will grow) were fixed by the maintainers.

9 Softirq context - the struggle continues

Having covered process and hardirq contexts we come to the stickiest problem - softirqs (aka "Bottom Halves", known as "DPCs" in the Windows world - all the work needed to handle an interrupt that can be delayed from the hardirq, and run later, on another processor, with interrupts enabled, etc). Full discussion of softirqs is outside the scope (see (Love, 2003)) of this paper but an important feature of the Linux implementation is that while softirqs normally run immediately after the hardirq that enabled them on the same processor in interrupt context, under load, all softirq handling can be offloaded to a "softirqd" thread, for scalability reasons.

An important side effect is that the kernel can be trivially modified to unconditionally run softirqs in process context, which results in a dramatic improvement in latency if the audio system runs at a higher priority than the softirq thread(s). This is the approach taken by the -rt kernel, and by many independent patches that preceded it.

The mainline Linux kernel lacks this feature, however, so minimizing scheduling latency requires limiting the amount of time spent in softirq context. Softirqs are used heavily by the networking system, for example looping over a list of packets delivered by the network adapter, as well as SCSI and for kernel timers (Love, 2003). Fortunately the Linux networking stack provides numerous sysctls that can be tuned to limit the number of packets processed at once, and the block IO fixes described elsewhere for IDE also apply to SCSI, which does IO completion in softirq context.

Softirqs are the main source of excessive scheduling latencies that, while rare, can still occur in the latest 2.6 kernel as of this writing (2.6.16-rc5). Timer based route cache flushing can still produce latencies over 10ms, and is the most problematic remaining softirq as no workaround seems to be available; however the problem is known by the kernel developers and a solution has been proposed (Dumazet, 2006).

10 Performance issues

The problems described so far mostly fit the pattern of too much work being done at once in some non-preemptible context and were solved by doing the same work in smaller units. However several areas where the kernel was simply inefficient were resolved, to the benefit of all users.

One such problem was kallsyms_lookup(), invoked in cases like printk(), which did a linear search over thousands of symbols, causing excessive scheduling latency. Paulo Marques solved this problem by rewriting kallsyms_lookup() to use a more efficient search algorithm. The frequent invocation of SHATransform() in non-preemptible contexts to add to the entropy pool was another latency problem solved by rewriting the code to be more efficient.

11 Non-kernel factors

The strangest latency problem identified was found to have an origin completely outside the kernel. Testing revealed that moving windows on the desktop reliably caused JACK to report excessive delays. This is a worse situation than an xrun as it indicates the audio device stopped producing/consuming data or a hardware level timing glitch occurred, while an xrun merely indicates that audio was available but JACK was not scheduled in time to process it. The problem disappeared when 2D acceleration was disabled in the X configuration which pointed clearly to the X display driver - on Linux all hardware access is normally mitigated by the kernel except 2D XAA acceleration by the X server.

The VIA Unichrome video card used in testing has a command FIFO and a status register. The status register tells the X server when the FIFO is ready to accept more data. (Jones and Regehr, 1999) describes certain Windows video drivers which improve benchmark scores by neglecting to check the status register before writing to the FIFO; the effect is to stall the CPU if the FIFO was full. The symptoms experienced were identical to (Jones and Regehr, 1999) - the machine stalled when the user dragged a window. Communication with the maintainer of the VIA unichrome driver (which had been supplied by the vendor) confirmed that the driver was in fact failing to check the status register and was easily fixed.

12 The -rt kernel and the future

The above solutions all have in common that they reduce scheduling latencies by minimizing the time the kernel spends with a spinlock held, with preemption manually disabled, and in hard and soft IRQ contexts, but do not change the kernels behavior regarding which contexts are preemptible. Modulo a few remaining, known bugs, this approach is capable of reducing the worst case scheduling latencies to the 1-2ms range, which is adequate for audio applications. Reducing latencies further required deep changes to the kernel and the rules about when preemption is allowed. The -rt kernel eliminates the spinlock problem by turning them into mutexes, the softirq by the softirq method previously described, and the hardirq issue by creating a set of kernel threads, one per interrupt line, and running all interrupt handlers in these threads. These changes result in a worst case scheduling latency close to 50 microseconds which approaches hardware limits.

13 Conclusions

One of the significant implications of the story of low latency in kernel 2.6 is that I believe it vindicates the controversial ”new kernel development process” (Corbet, 2004) - it is hard to imagine Linux 2.6 evolving into a world class audio platform as rapidly and successfully as it did under a development model that valued stability over progress. Another lesson is that in operating systems as in life, history repeats itself. Much of the work done on Linux 2.6 to support soft realtime applications, like IRQ threading, was pioneered by Solaris engineers in the early 1990s (Vahalia, 1996).

14 Acknowledgements

My thanks go to Ingo Molnar, Paul Davis, Andrew Morton, Linus Torvalds, Florian Schmidt, and everyone who helped to evolve Linux 2.6 into a world class realtime audio platform.

References

Jonathan Corbet. 2004. Another look at the new development model. https://lwn.net/Articles/95312/.

Eric Dumazet. 2006. Re: Rcu latency regression in 2.6.16-rc1. http://lkml.org/lkml/2006/1/28/111.

Michael B. Jones and John Regehr. 1999. The problems you’re having may not be the problems you think you’re having: Results from a latency study of windows nt. In *Proceedings of the 7th Workshop on Hot Topics in Operating Systems (HotOS VII)*, pages 96–101.

Robert Love. 2003. *Linux Kernel Development*. Sams Publishing, Indianapolis, Indiana.

Ingo Molnar. 2004. [announce] [patch] voluntary kernel preemption patch. http://lkml.org/lkml/2004/7/9/138.

Uresh Vahalia. 1996. *Unix Internals: The New Frontiers*. Prentice Hall, Upper Saddle River, New Jersey.