LOW-LATENCY MUSIC SOFTWARE USING OFF-THE-SHELF OPERATING SYSTEMS

Eli Brandt and Roger Dannenberg

School of Computer Science, Carnegie Mellon University
Pittsburgh, PA 15213 USA
{rbd, eli}@cs.cmu.edu

Abstract: Operating systems are often the limiting factor in creating low-latency interactive computer music systems. Real-time music applications require operating system support for memory management, process scheduling, media I/O, and general development, including debugging. We present performance measurements for some current operating systems, including NT4, Windows95, and Irix 6.4. While Irix was found to give rather good real-time performance, NT4 and Windows95 suffer from both process scheduling delays and high audio output latency. The addition of WDM Streaming to NT and Windows offers some promise of lower latency, but WDM Streaming may actually make performance worse by circumventing priority-based scheduling.

1. Introduction

A modern CPU can run extensive and sophisticated signal processing faster than real time. Unfortunately, executing the right code at the right time and performing time-critical I/O may very well exceed the capabilities of a modern operating system. Latency is the stumbling block for music software, and the OS is often the cause of it.

After looking at latency and other requirements for music software, we consider the support provided by the operating system. We survey current operating systems, discussing their capabilities and performance and our experience with software development. Finally, we consider some directions for future operating systems.

2. What do we want from music software?

Every piece of data passing through an application will be delayed from input to output. A data path's latency may be constant, or it may vary, in which case both the worst-case latency and the latency distribution's spread—the jitter—are perceptually significant.

An interactive application has certain latency constraints on each path through it. MIDI rendered to audio, audio rendered to a spectrogram, mouse modulating audio mix—the constraints vary by application. End-to-end latency depends on the latencies of input, processing, and output. There do not seem to be published studies of tolerable delay, but our personal experience and actual measurements of commercial synthesizer delays indicate that 5 or maybe 10 ms is acceptable. This is comparable to common acoustic-transmission delays; sound travels at about 1 foot/ms. As for jitter, the just-noticeable difference for a 100-ms rhythmic period has been reported as 5 ms, 1 ms, and 3 or 4 ms (Van Noorden 1975, Michon 1964, and Lunney 1974).

3. What does music software want from the OS?

To perform I/O and to process data in real time, an application needs timing guarantees from the operating system. Typically, I/O is performed by a combination of hardware and software within the OS. The application program is either notified by the OS, or the application queries the OS to determine that it is time to process more data. Regardless of the details, the application must run promptly to meet real-time requirements.

The most stringent requirement will normally be on audio output because an audible click or pop results if the buffer underflows. An ideal configuration might be: Compute a 1-ms block of audio every ms. At the beginning of each block computation, have 2 ms already in the output buffer. (This allows 1 ms of computation time and 1 ms of worst-case scheduling latency before the output buffer empties.) The worst-case audio latency would then be 3 ms. An event arriving just after the start of a block computation affects the audio stream starting on the next block, the first sample of which will be output in 3 ms.

Unfortunately, many things can prevent an application program from processing data in time. One example is that the application's code or data may have been paged out to disk, so it must wait for the memory to be restored. Most operating systems with virtual memory can "pin" memory to prevent paging, benefiting tasks that cannot afford to wait for a disk. However, identifying all the memory pages to pin is not simple when the application needs extensive libraries and other subsystems.

A ready process or thread will eventually be scheduled to run by the OS. The crucial latency from "ready" to "running" is determined by what other ready processes run first. The programmer needs to be able to influence the scheduler to run everything on time, so an OS should recognize sufficient priorities to distinguish among the application's tasks. These priorities should be fixed, with no attempt to provide long-term fairness to low-priority processes. An inheritance scheme (Sha, Rajkumar, and Lehoczky 1990) to handle priority inversion is desirable, particularly if the OS itself employs locks that may cause one system call to unexpectedly block another. More sophisticated scheduling is also possible.

Perhaps it goes without saying that music software demands access to musical media like MIDI and audio, as well as to disk, network, the console, and other devices. The presence and quality of drivers and APIs, while unglamorous, is influential in selecting an OS. Given privileged access to the hardware, an application can also address it directly, but this limits portability and interoperability; device drivers really are best written by the device manufacturer, once only.

Another I/O issue, especially for MIDI, is timestamping. An incoming message may wait for some time before the application deals with it. Without a timestamp, the handler's scheduling latency is propagated to the output. A timestamp may enable the handler to hide jitter by adding a deliberate delay to achieve a constant latency. Similarly, the application might send its MIDI output a bit early, timestamped for output at precisely the right time [Anderson and Kuivila 1986].

Finally, one must consider the type of program-development tools that are available. The weakest link is often the symbolic debugger, without which certain bugs are quite difficult to track down. This is not strictly an OS issue, but good debugging support for application development in mainstream operating systems is an important consideration.

4. What can software get from today's operating systems?

We consider Windows NT 4 SP3, Windows 95, and Irix 6.4. While it might seem only sensible to develop one's real-time software for a *real* real-time operating system, a specialized one, we exclude them from this survey. Often they lack drivers for the devices we need, especially state-of-the-art devices for audio I/O and computer animation. In addition, research software is of limited value if people do not have or cannot afford the OS it runs under. There are attractive options, including BeOS, a commercial "media OS", and rtLinux, a free real-time kernel—measured at 64-µsec scheduling latency [Barabanov 1997]—running Linux as a subsystem. With more users and driver support from hardware manufacturers, these may become more attractive.

One could take NT4 as the model of a modern general-purpose OS. NT is not designed for real-time music, but it will suffice for some applications. NT4 has drivers for the essential hardware, and usable APIs. For MIDI, events are not timestamped, and input notification can be through polling, a callback, a thread, or a semaphore. Audio is multiple-buffered; the application specifies the number and size of buffers, although this may or may not affect the driver's internal buffering. The notification options also include signaling a semphore.

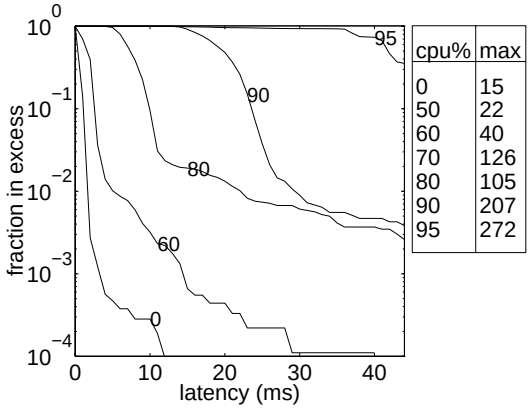
The NT implementation of Aura [Dannenberg and Brandt 1996] illustrates NT's capabilities. Briefly, Aura is a software synthesis architecture based on a real-time object system [Dannenberg 1995]. Objects are partitioned among single-threaded *zones*; intra-zone messages are synchronous and inter-zone messages are asynchronous. In order of decreasing priority, zones are generally for audio, MIDI, and the user interface. Within the audio zone, specialized *instrument* objects compute data on demand; their protocol is lightweight to allow computation that is fine-grained and dynamically configurable, yet efficient. NT's audio subsystem (including drivers) is presently the factor determining Aura's 186-ms audio output latency. DirectSound is an alternative API with single buffering, polled or signaled. It should offer lower latency, but in NT4, DirectSound is emulated on top of the old API.

A modified version of Microsoft's LATENCY code uses the same priority and the same timer as Aura, and measures the actual time between nominal 5-ms callbacks. This can be used to determine the latency NT applications can expect. The results of running this test are shown in Figure 1, a log plot of the fraction of callbacks that exceed a given latency shown on the horizontal axis. Different curves correspond to different levels of contention for the machine.

Tests were run for one minute on a newly-booted IBM PC/365 (200-MHz Pentium Pro, 64M RAM), except Windows 95 on a ThinkPad 760 (166-MHz Pentium, 40M RAM). For NT, all daemons except EventLog and RPC were stopped. Load was a "find in files" and Microsoft's `cpugrab` utility (at normal priority). Disabling the network (IBM PCI Token Ring driver v3.34.00.0040) makes no discernable difference. Dragging or manipulating windows during the test readily causes quarter-second delays, so this was avoided. Although the contending

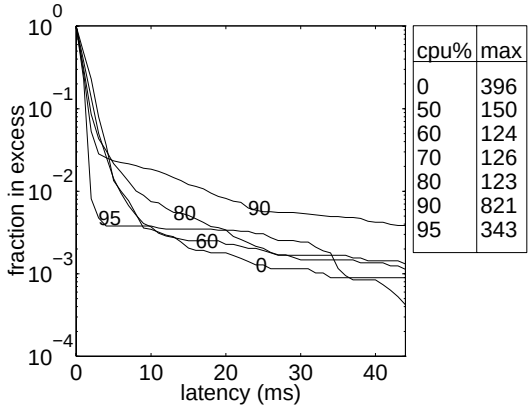
`cpugrab` process is at normal priority, at higher settings it appears more demanding than the typical CPU-bound process.

Average excess latency increases with load, and worst-case latency is uniformly quite high. With full CPU load, but the network unplugged, Dannenberg and Goto (1997) report 20 ms worst-case, which we see here at 50% load.



cpu%	max
0	15
50	22
60	40
70	126
80	105
90	207
95	272

Figure 1: fraction of 5-ms timer callbacks exceeding a given latency under NT4, with various CPU loads; table of worst-case latencies.



cpu%	max
0	396
50	150
60	124
70	126
80	123
90	821
95	343

Figure 2: fraction of 5-ms timer callbacks exceeding a given latency under Win95, with various CPU loads; table of worst-case latencies.

One might try to improve on this by running as a device driver, above the scheduler's domain, preferably by leaving NT4 and writing a WDM Streaming driver, as discussed below. This complicates development somewhat, and although NT kernel debugging is better than most, it does involve a "checked" OS build and a second machine. Alternatively, one might use an NT-hosted real-time kernel. In cost, user base, and hardware support, these are like the stand-alone RTOS's we ruled out above, but they also offer solid real-time performance. Using Imagination Systems' Hyperkernel, we measured worst-case scheduling latency under 35 µs. This approach permits non-real-time use of NT's facilities, and can involve the same development tools, debugger excluded.

Real-time programming under MacOS or Windows 3.1 is something like device-driver programming: the code runs in an interrupt handler, without preemptive multitasking or priority-based scheduling. The results can be good—about 5 ms worst-case scheduling latency under MacOS on a 60-MHz PPC—but the programming model may be limiting.

Windows 95 is similar to NT4 in many respects. It does have native DirectSound, however. A trick not applicable in NT is that of taking multimedia timer callbacks into a 16-bit section of code. This improves timing substantially, for example from 128 to 23 ms worst-case under 80 percent load. Another option is to place real-time code in a device driver. Figure 2 shows latency for a 32-bit timer. Notice that Win95's latency is less sensitive to load than NT's; it does not degrade nearly so dramatically, but does have uniformly poor worst-case latency.

SGI's Irix has good MIDI and audio support and offers relatively reliable scheduling. Thus our port of Aura to Irix 6.4 achieved 7 ms audio output latency in preliminary testing. This is markedly superior to the NT4 version. Rather than buffering in units of large blocks, the Irix audio API can transfer small numbers of sample frames. Irix programs can use the Unix `select()` call for all notification and timing.

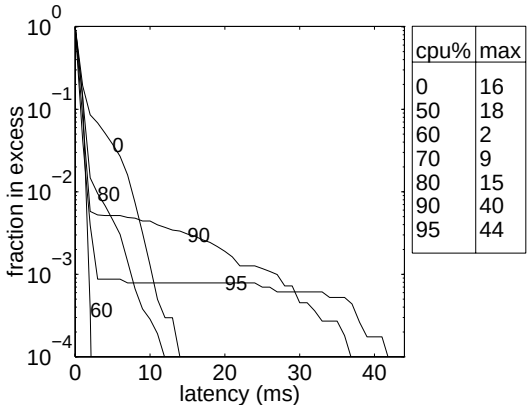
To get Irix's process rather than thread scheduler, we implemented zones as processes sharing an address space. Their priorities were set to 150 and up, within the real-time priority range but below kernel operations. Under light CPU load, the audio zone's nominal 2 ms timeout showed 3 ms worst-case scheduling latency,[1] whereas Dannenberg and Goto saw 5 ms under more strenuous conditions. These settings consumed 7 percent of the CPU as polling overhead. Further Irix experience is reported by Freed (1997).

[1] Run on an SGI Octane (one R10000 processor); load consisted of grep'ing all files on the disk and starting many Netscapes.

5. What can software expect from tomorrow's operating systems?

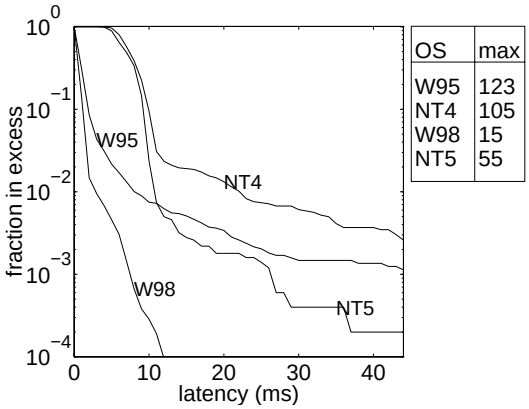
Apple has announced MacOS X, to be based on the same monolithic-kernel Mach 2.5 architecture as NextStep. As this kernel's real-time limitations [Nakajima91] have not been addressed during its use in NextStep, they seem likely to persist into MacOS X.

Windows 98 was recently released, and NT5 will be available in the foreseeable future. As seen in Figure 3, the scheduling latency of pre-release versions seems to be superior to that of each system's predecessor, especially for Windows 98. NT5 will have native DirectSound, which should improve its audio behavior. It also allows floating point in the kernel.



cpu%	max
0	16
50	18
60	2
70	9
80	15
90	40
95	44

Figure 3: fraction of 5-ms timer callbacks exceeding a given latency under Win98 Release Candidate 0 , with various CPU loads; table of worst-case latencies.



OS	max
W95	123
NT4	105
W98	15
NT5	55

Figure 4: fraction of 5-ms timer callbacks exceeding a given latency under various operating systems, with 80% CPU load; table of worst-case latencies.

Each adds support for WDM, the Windows Device Model. Aside from presenting a unified target for device-driver development (ending NT's drought of audio-hardware drivers), this includes WDM Streaming, a new scheme for low-latency media processing.

WDM Streaming works around the scheduler by running processing code in the kernel. Both it and DirectShow, the user-mode framework for not-quite-so-low-latency media processing, build directed graphs of processing modules ("filters"), and stream data through from sources to renderers. A wide variety of media types are defined, including audio (and WAVE_FORMAT_FLOAT, finally) and to some extent MIDI. WDMS and DirectShow share media types and streaming semantics, but WDMS filters are WDM device drivers, while DirectShow uses user-mode COM objects.

A device driver may have an interrupt service routine (ISR), which runs at a high IRQ level, i.e. with lower-priority interrupts masked. The ISR, to minimize its execution time, typically enqueues a deferred procedure call (DPC) for any substantial work to be done. DPCs run in the kernel at an IRQ level below those of ISRs and above that at which user code is scheduled. Floating point will be permitted at the DPC level. WDMS filters will do the bulk of their processing in DPCs.

We do not have timing measurements for DPCs; they would be of little use anyway. DPCs are scheduled using a FIFO discipline and run to completion. A device driver running alone in the kernel could guarantee that its DPCs run on time by constraining their completion times. But in the presence of arbitrary DPCs, the system makes no guarantee as to when any DPC will run. Furthermore, since DPCs run before processes, the worst-case process latency is at least as long as the longest-running DPC. Thus, the real-time performance of the system can be thwarted by an uncooperative device driver.

This problem is not new to NT device-driver developers, but WDMS introduces it to a wider pool of media programmers. They (we) might benefit by trying to develop guidelines for writing interoperable software, or for analyzing and documenting DPC timing.

Another interesting question is what the protocol overhead will prove to be for fine-grained computation. DirectShow includes timestamped chunks of data, breaks in the stream, and in-band format changes, involving a fairly elaborate protocol. On a 200-MHz Pentium Pro, we took DirectShow as implemented by the DirectX Foundation 5.2 SDK for NT4, and timed an optimized in-place transform filter doing nothing: it takes 3.3 μs. As a

naive memory-to-memory multiply takes about 3.7 μs for a 32-sample block of four-byte floats, the DirectShow overhead for the simplest 32-sample block would be about 90 percent. For comparison, Aura's per-instrument block overhead is 0.08 μs, or 2 percent here.

This DirectShow measurement may not extend to WDMS: the documentation presently released is not comprehensive on WDMS's use of DirectShow streaming semantics. And of course this is an extreme; the overhead will be proportionally less on a more time-consuming block.

6. Summary and Conclusion

The ideal system for interactive music programming would combine the development tools, languages, device support, and user base of a consumer-oriented OS with the hard real-time guarantees of a real-time OS. Irix has good real-time performance and supports a variety of audio, MIDI, and graphics interfaces, making it a capable platform. For ordinary code running as a user-level process, NT4 is limited by its audio system, and Window 95 by its scheduler. NT5's improved audio may merely unmask scheduler limitations, but it appears that Windows 98 could achieve reliable 20-ms audio output latency.

WDM Streaming is an attractive new interface for implementing low-latency streaming. However, combinations of applications may fail to work together, as it exposes them directly to one another's timing.

7. Acknowledgements

The first author was funded by the IBM University Partnership Program and by Interstate Electronics Corporation, and would also like to thank the IBM Computer Music Center for their support and the use of their facilities.

References

Anderson, D. P. and R. Kuivila. 1986. "Accurately Timed Generation of Discrete Musical Events." *Computer Music Journal* 10(3): 49–56.

Barabanov, M. 1997. "A Linux-based Real-Time Operating System." MS thesis. New Mexico Institute of Mining and Technology, Socorro, New Mexico.

Dannenberg, R. B. and D. Rubine. 1995. "Toward Modular, Portable, Real-Time Software." In *Proceedings of the 1995 International Computer Music Conference, International Computer Music Association*, pp. 65–72.

Dannenberg, R. B. and M. Goto. 1997. "Latency in Computer Audio Systems", *Array Online*. http://music.dartmouth.edu/~icma/array/spring97/articles.html.

Dannenberg, R. B. and E. Brandt. 1996. "A Flexible Real-Time Software Synthesis System." In *Proceedings of the 1996 International Computer Music Conference*, International Computer Music Association, pp. 270-273.

Freed, A., A. Chaudhary, and B. Davila. 1997. "Operating Systems Latency Measurement and Analysis for Sound Synthesis and Processing Applications." In *Proceedings of the 1997 International Computer Music Conference*, International Computer Music Association.

Lunney, H. M. W. 1974. "Time as heard in speech and music." *Nature* 249, p. 592.

Michon, J. A. 1964. "Studies on subjective duration 1. Differential sensitivity on the perception of repeated temporal intervals." *Acta Psychologica* 22, pp. 441–450.

Nakajima, J., M. Yazaki, and H. Matsumoto. "Multimedia/Realtime Extensions for the Mach Operating System." In *Proc. Summer USENIX Conf.*, pp. 183–198. USENIX Association, 1991.

Sha, L., R. Rajkumar, and J. P. Lehoczky. 1990. "Priority Inheritance Protocols: An Approach to Real-Time Synchronization." In *IEEE Transactions on Computers*, vol. 39, pp. 1175–1185.

Van Noorden, L. P. A. S. 1975. *Temporal coherence in the perception of tone sequences*. Unpublished doctoral thesis. Technische Hogeschool, Eindehoven, Holland.