TIMING MEASUREMENTS IN JACK2

Stéphane Letz, Dominique Fober, Yann Orlarey

TIMING MEASUREMENTS IN JACK2

S.Letz, D.Fober, Y.Orlarey
Grame - Centre national de création musicale
letz, fober, orlarey@grame.fr

ABSTRACT

Timing measurements allow developers to help understanding the behaviour of their JACK based applications. The server code base now allows to record various timing while the server and clients are running and generate scripts to interpret them.

1. INTRODUCTION

When porting JACK2 code base on a new operating system, it is often necessary to get a precise knowledge of the host operating system real-time capabilities. JACK2 now contains additionnal code that allows to have a better understanding of the timing behaviour of the server and all running clients.

2. WHAT IS MEASURED

To activate profiling, the JACK server has to be recompiled with special flags. Use the –profile at configure step: **./waf configure –profile**. While running, timestamps are taken for the server and running clients: *wake-up date* of audio driver, *activation*, *actual wake-up* and *end date* of each running client. Client scheduling latency and duration is also computed. They are saved as a *JackEngineProfiling.log* file containing time points when the server is closed. [1] Scripts for Gnuplot are also generated. Use the command: **gnuplot -persist Timing1.plot** with Timing1.plot up to Timing5.plot. The scripts also generate PDF files. A **jack_profiler** internal client allows to see all measures as *audio signals* to be analyzed or recorded with additional tools.

2.1. Audio driver interrupts

Timing1.plot allows to display the audio driver timing, that is the duration between consecutives interrupts.

When the interrupt period is regular, then the server *asynchronous* mode can be safely used (fig 1). On the contrary a non regular driver interrupt force to *synchronous* mode to be chosen (otherwise the graph may lack time to finish its execution if duration between 2 consecutive interrupts is too short) (fig 2).

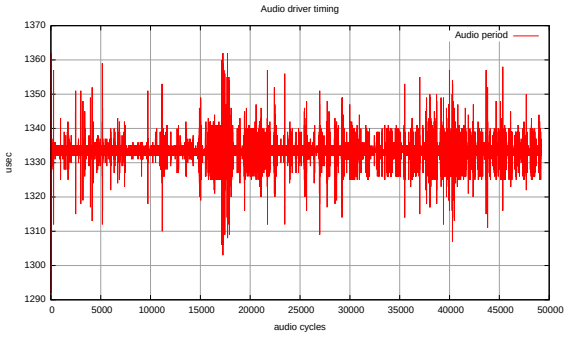

Figure 1. *Audio driver interrupt, at 48 kHz and 64 frames, average is 1333 usec, interrupt period is regular (with small variations of about +/- 30 usec)*

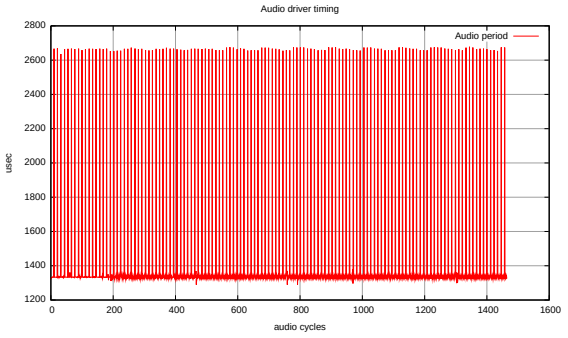

Figure 2. *Audio driver interrupt, at 44.1 kHz and 64 frames, average is 1451 usec, but interrupt period is not regular*

2.2. Driver end date

Timing2.plot allows to display the audio driver end date. This measure is interesting to distinguish what happens in server synchronous versus asynchronous mode. In synchronous mode, the audio cycle is composed of: *read audio input buffers, execute the graph, write audio output buffers*. Thus the driver end date takes the graph execution duration in account (fig 3). [2] In asynchronous mode on the contrary, the audio cycle is composed of: *read audio input buffers, write audio output buffers computed at previous cycle, execute the graph* (fig 4). The driver does

[1] The server has to be closed first to correctly retrieve informations about the running clients.

[2] The graph here shows several clients launched one by one, the driver end date then raise after each new client is started.

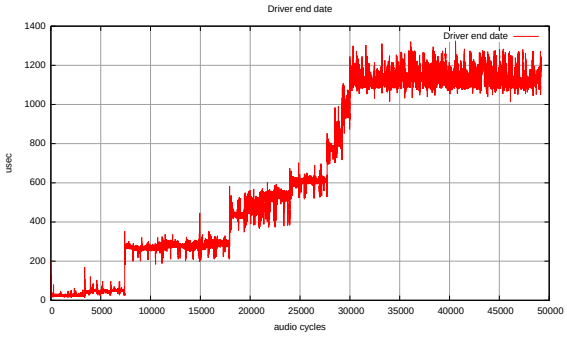

Figure 3. *Driver end date when the server runs in synchronous mode*

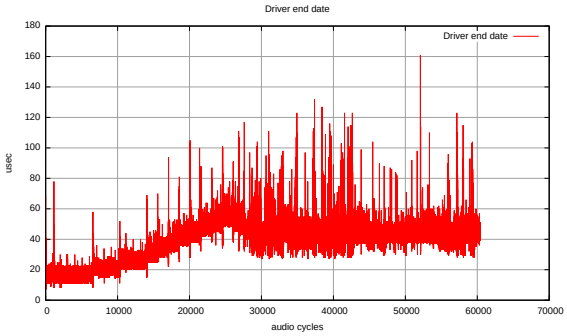

Figure 4. *Driver end date when the server runs in asynchronous mode*

not wait for the graph end, but returns immediately after the write step. Thus the driver end date measures the *read audio input buffers, write audio output buffers* parts only.

2.3. Clients end date

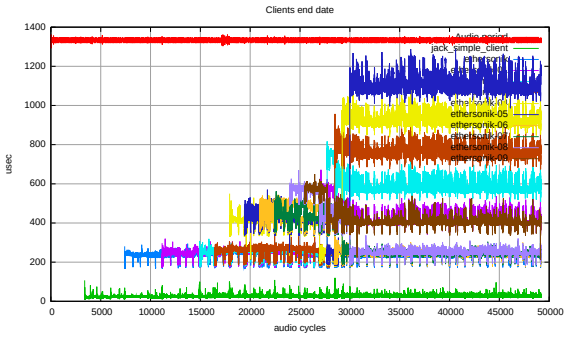

Figure 5. *Clients end date*

Timing3.plot allows to display the audio driver timing and all clients end date. This curve gives a global view of all clients DSP use. The audio interrupt duration is displayed as well as the end date of all running clients. The system works correctly if the end date of the last client is still below the audio interrupt duration (fig 5).

2.4. Clients scheduling latency

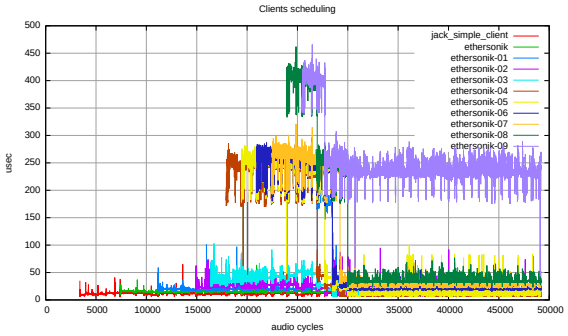

Figure 6. *Clients scheduling latency*

Timing4.plot allows to display all client scheduling latencies (difference between activation and actual wake-up dates). When the application real-time thread becomes runnable, the global scheduling latency depends on the fact a core is effectively available in the machine and the actual OS scheduling latency. Thus this value obviously depends of the topology of the graph and number of available cores on the machines on a given setup. To precisely measure the OS scheduling latency only, the best is to have a number of clients that is less than the number of available cores (fig 6).

2.5. Clients duration

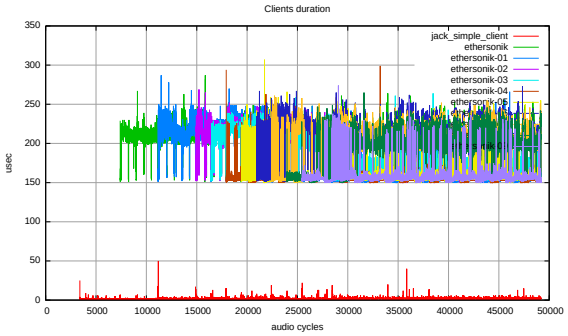

Figure 7. *Clients duration*

Timing5.plot allows to display all client duration (difference between end date and actual wake-up date) (fig 7).

3. REAL-TIME MEASUREMENTS

The **jack_profiler** internal client allows to see all measures as *audio signals*. It can be loaded at anytime using the jack_load tool. JACK output ports will be created and contain timing measures converted in audio signals. At each audio cycle, the measured value is re-calibrated to be in the [-1,1] range and copied into the audio buffer (same

value for the entire cycle). Three general output JACK ports can be added (respectively using the *-c, -p, -e* parameters in the jack_load parameter line):

- **profiler:cpu_load**: is the DSP CPU load between 0 and 1.
- **profiler:driver_period**: is the driver period variation expressed at the difference between the actual driver period and the expected driver period, then divided by the expected driver period, between -1 and 1.
- **profiler:driver_end_time**: is the driver end time expressed as driver end time divided by the driver period, between 0 and 1.

For each running client, two additional JACK ports will appear:

- **profiler:"client_name":scheduling**: is the client scheduling duration expressed as the scheduling duration divided by the driver period, between 0 and 1.
- **profiler:"client_name":duration**: is the client duration expressed as the client duration divided by the driver period, between 0 and 1.

Measured signals can then be analyzed using additional real-time displaying tools, or possibly recorded with any recording application. It could be even possible to **sonify** them to detect problematic events by ear.

4. RESULTS ON SOLARIS

The timing measurements tools have been developed in the context of JACK2 port to Solaris. The set of curves in this paper have been done on a Dell XPS 420 Intel 4 cores machine running Solaris 10 and using the internal HD Audio card using OSS 4.0 version. Fig 1 shows the audio driver interrupts when the server is running using a 64 frames buffer size at 48 kHz, in synchronous mode and using the highest scheduling priority that can be obtained (using the following command: **jackd -R -S -P 59 -d oss -p 64).** Then fig 2 to 5 shows the different curves obtained with a highly loaded machine.

4.1. Using Processor Sets

More tests have been done to evaluate the effect of Processor Sets. A *Processor Set* is a group of processors on which processes can be forced to run. Moreover *Interrupt Sheltering* allows to shelter a processor set from unbound interrupts. A first measure has been done with 3 clients consuming something like 20% of the available DSP CPU on a 30 min period (that is 3 clients using 20% DSP CPU each but running in parallel) with the machine doing a multi-threaded compilation at the same time. The maximum scheduling latency is about 110 usec (fig 8).

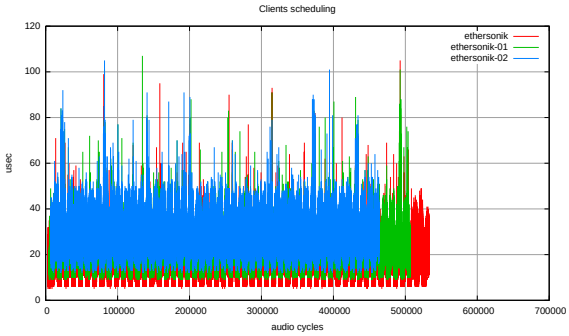

Figure 8. *Clients scheduling latency*

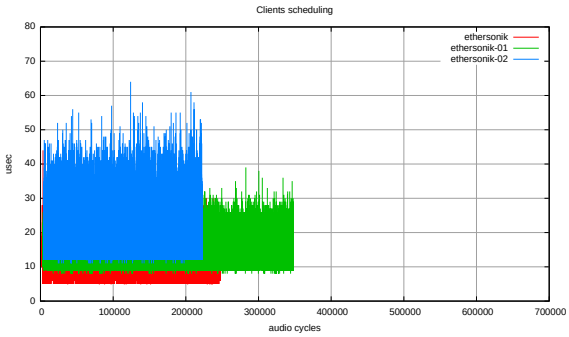

Figure 9. *Clients scheduling latency: processor set*

Next a processor set is defined with **psrset -c 1 2 3** to set processors 1,2,3 in the processor set 1 (the returned processor set index, here 1, is used in further commands). Then **psrset -f 1** command puts all processors in the specified processor set in the P_NOINTR state (that is processor will be available for thread scheduling, but does not handle network or I/O interrupts and processor 0 is kept unsheltered to service the system clock interrupt). Processor state can be checked using the **psrinfo** command and the binding of LW threads in the processor set can be checked using **psrset -Q 1**. Any program can be started on the given processor set using **psrset -e 1 command**. The JACK server and 3 clients have been started this way: **psrset -e 1 jackd -R -S -P 59 -d oss -p 128** and **psrset -e 1 ethersonik** for each 3 client. [3] The maximum scheduling latency is now lowered to about 80 usec (fig 9).

Still using a processor set, another measure is done with the machine only loaded with JACK applications (no more compilation). The average scheduling latency is now lowered to about 35 usec (fig 10), but they are some scheduling latency peaks when each application starts that probably correspond to *application code warming*.

[3] Real-Time access for a user can be given using the **priocntl -s -c RT -i uid num** command where num is the user uid, then all further commands will benefit of this capability.

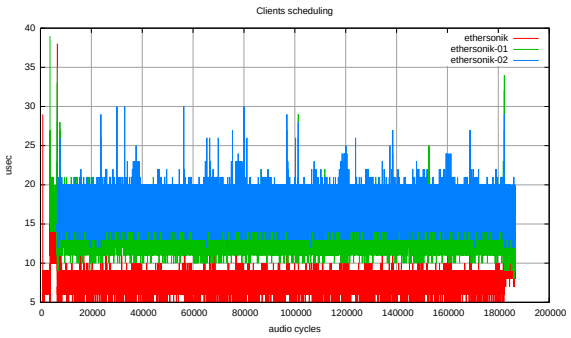

Figure 10. *Clients scheduling latency: warming code effect*

4.2. Parameters setting on Solaris

To obtain the best possible behaviour of the system, the following points have to be taken in account:

- It appears that the OSS driver does not guaranty a perfectly regular audio interrupt period, depending of the driver settings. Thus the *synchronous* (-S) mode should be preferably chosen when starting the server.
- Solaris defines several scheduling classes. The Real-Time scheduling class goes from 100 to 159 on a global 0 to 169 scale [2]. The corresponding POSIX priority to be set in JACK will go from 0 to 59 (-P59 for highest priority).
- The Solaris has quite good Real-Time capabilities, but as explained before, using Processor sets allows to decrease the RT threads scheduling latencies and have a more predictable system. We measured a maximum scheduling latency of 80 usec in this configuration.

5. CONCLUSION

Timing measurements are definitively of great help to understand the timing behaviour of a complex system using several JACK audio applications. It allows to clearly understand where abnormal latencies are coming from. More adapted tools like DTrace on Solaris would have to be used to better understand the reasons. Developed in the context of JACK2 port on Solaris, the tools show that the Solaris system has quite good real-time capabilities. This work has been funded by RTL french radio (EDIRADIO).

6. REFERENCES

[1] Gnuplot homepage, *http://www.gnuplot.info*

[2] S.Letz, D.Fober, Y.Orlarey, ”Jack audio server for multi-processor machines, *Proceedings of the International Computer Music Conference ICMA 2005 Pages 1–4*

[3] J.Litchfield, ”Real-time in the Operating Environment Solaris 8”, *http://www.opengroup.org/rtforum/oct2000 /slides/litchfield.pdf*

[4] K.Vehmanen, A.Wingo and P.Davis *Jack Design Documentation* http://jackit.sourceforge.net/docs/design/