Slackware Live DAW: Minimizing Latency

When is Minimal Latency Actually Needed?

Low latency in a digital audio workstation is needed when providing live playback to the artist. If audio coming from the monitor is delayed from the action of playing an instrument or singing, it interferes with the artist's ability to keep in time. This applies to recording a musician or vocalist with live monitoring, playing a live show with software instruments, mixing a live show through a DAW, or any other work where audio is being created while listening to it.

The threshold for latency being audible is around 10 msec (from playing until hearing through the monitor), and can also depend on the instrument played. A decent audio interface can be pushed below 10 msec round trip latency, whether its PCI, Firewire, or USB.

In other cases like editing and mixing tracks, the latency doesn't matter as much and it can be compensated easily. It is better to run the audio server with a larger buffer size, reducing the demand on the system. This allows you to work with more plugins, instruments, tracks, etc. and have a much lower risk of hitting buffer xruns.

This article assumes that you are interested in reducing latency in the Slackware Live DAW for the purpose of live playing, and will provide some ideas on how to achieve xrun free and stable operation at these demanding settings.

Slackware Live DAW as a Starting Point

The Slackware Live DAW Edition is a pre-configured Slackware distribution that has been optimized for audio work. Eric (aka AlienBOB) has already set up the typical prerequisites for real-time audio work, including:

- A real-time configured kernel (using the `'threadirqs'` boot option)
- CPU governor set to maximum performance
- Real-time priority scheduling and unlocked memory for the audio group (using PAM)
- HPET and RTC access set up with udev rules
- sysctl tweaks
- KDE Plasma 5 GUI tweaks

As a live distribution, getting the system set up just involves plugging in a USB stick and rebooting. There's no need to maintain two separate installs, or risk running real-time privileged applications on your day to day machine.

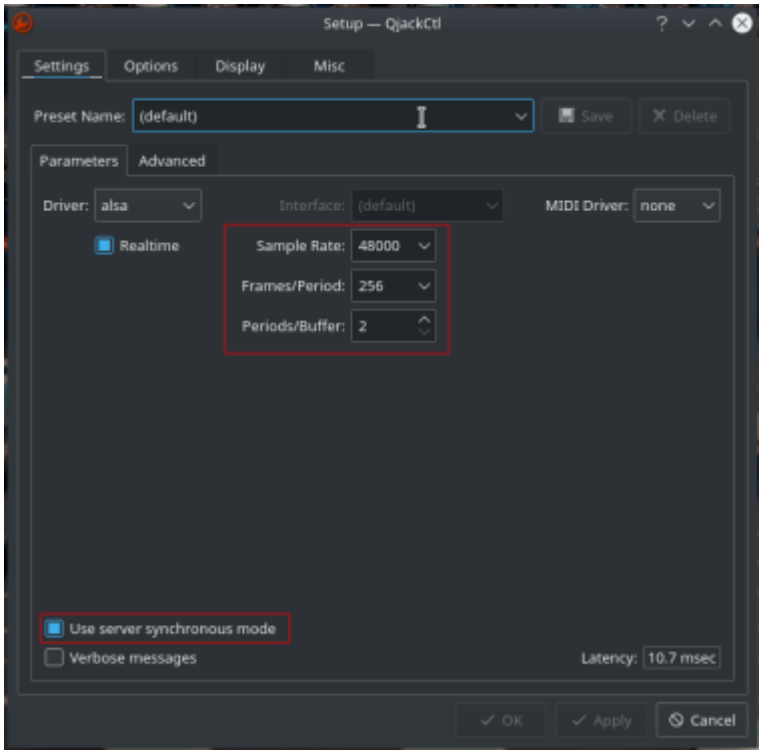
Of course the performance of the system is also dependent on the user's hardware. A USB, Firewire, or PCI audio interface makes all the difference in getting xrun free, low latency audio.

By default, the Slackware Live DAW will boot using the JACK audio server, managed by QJackCtl. The default settings are already lowered for real-time work and offers 10.7 msec of “calculated buffer latency”, which gives anywhere between 10msec - 20msec round trip latency when including the hardware's delay. The buffer settings can be lowered further, but the limit on how far depends on the

hardware. PCI devices generally achieve the lowest latency, although USB and Firewire can still work well.

Audio Server Settings

The key settings that will adjust the latency of the system for the audio server are as follows:



Sample Rate

This sets the speed of how many audio samples (called frames) are taken per second. A higher sample rate will be fill up the buffer faster, leading to lower latency. The sample rate of 48000Hz is default. Other typical sample rates are 44100Hz, 88200Hz, 96000Hz, and 192000Hz. Note that higher sample rates will load the CPU more than lower rates.

Frames/Period

This setting determines the size of the buffer, in conjunction with the Periods/Buffer setting. The default settings use base 2 numbers (i.e 16, 32, 64, 128, 256, etc.), although custom values can be entered. Just be aware that some plugins are hard-coded to only work with base 2 values for Frames/Period, and will throw errors if set to anything otherwise. E.g. guitarix's cabinet simulator wont work and gives errors if the audio server isn't running a base 2 frames/period setting.

Custom values come in useful for some USB interfaces, which perform better with a buffer latency that is a whole number multiple of milliseconds, For example, a setting of 48000Hz, 48 frames/period, and 2 periods/buffer will use a buffer of 48 x 2 = 96 samples, and at 48000Hz, this takes 96/48000 =

2 milliseconds to fill. Applying this logic, the optimal settings for USB devices have been documented here: https://wiki.linuxaudio.org/wiki/list_of_jack_frame_period_settings_ideal_for_usb_interface

Lower size buffers require the CPU to work harder to keep up with the sample rate.

Periods/Buffer

This setting works with the Frames/Period to set the size of the buffer. Values of either 2 or 3 are used here. The man page for jackd recommends using 3 for USB and Firewire interfaces, and 2 for everything else (note: that is just a recommendation, not a rule). Values of 3 provide more combinations for whole numbers of millisecond buffer latency, but there are several combinations with 2 that also work. E.g. 48000Hz/48/2 can be used with a USB interface, providing the computer is sufficient for the task. A table of optimal USB interface settings can be found at the link in the previous section, or the references of this article.

Synchronous Mode

In synchronous mode a cycle of the JACK server reads the inputs, executes the graph, and writes the outputs. In asynchronous mode the server doesn't wait for the graph to execute, writing the outputs in the next cycle. The end result is that asynchronous mode adds an extra period to the buffer, making the buffer latency of asynchronous mode slightly higher than synchronous mode.

Testing Audio Server Settings

Having a variety of setting options leads to a number of possible combinations that can work with the audio server. When testing out different settings to find what works best, examine the DSP load when you start the server at the new settings. DSP load will be low at server startup, since there is no load yet. On a well configured machine DSP load will be under 1% while idle. Running applications and plugins increases the load, and at a high enough load the system will xrun. If the DSP load is higher at no-load, that means you can expect less plugins and higher chances for xruns for that setting.

A program called 'xruncounter' was developed by the linuxmusicians.com community that can be used to more thoroughly test a JACK audio server. After starting the server, '`xruncounter`' is run from a terminal. (note: '`xruncounter -s`' runs a multicore test), which then builds up DSP load over time, and exits automatically once it detects xruns. This provides an indication of how well the system will perform under load for a given setting. A well configured system will get into the high 90% range of DSP load before hitting xruns.

The xruncounter program also provides an indication of the amount of programs or plugins from the cycle count hit by the first xrun. A higher number of cycles before xruns occur indicate that more plugins can be used at that setting. The best settings will run up to a high DSP load, have a higher cycle count, and still have acceptable round trip latency.

The following image shows an example of xruncounter's output, with good results:

```
Samplerate is 48000Hz
Buffersize is 64
Buffer/Periods  3
jack running with realtime priority 85
Xrun 1 at DSP load 99.82% use 1.33ms from 1.33ms jack cycle time

in complete 1 Xruns in 35629 cycles
first Xrun happen at DSP load 99.82% in cycle 35530
process takes 1.33ms from total 1.33ms jack cycle time
```

Further Tuning Options

There are a number of options that can be taken to increase your hardware's stability at low latency settings. In general, these measures are taken to either reduce DSP load, or to increase stability of the computer, since this is where we can make changes. In no particular order, some things that can help are:

Limit CPU C-States to C1

CPU C-states are the various levels of power saving modes that the CPU can enter when idling. C-states are useful for day to day computer usage, since they reduce energy consumption. However, the time taken for the CPU to switch in and out of higher C-states can reach into the millisecond range, which can be longer than the audio server's buffer time at low latency settings. This will manifest itself as occasional xruns occurring on the system, even when the DSP load is low. Limiting C-states to C1 will keep the CPU cores running at full speed, and will remove these xruns, at the expense of a slightly warmer CPU.

CPU clock frequency will also run more consistently when limited to C1, which gives good stability and xrun free audio for longer live sessions. CPU clock speed can be monitored from a terminal with a command like “`watch -n 0.1 grep MHz /proc/cpuinfo`”. When limited to C1, the frequency reported here will not vary beyond a 1 MHz difference. Without the limit the CPU frequency will bounce around more, which can cause occasional xruns.

C-states can be limited to C1 by booting with `'intel_idle.max_cstate=1'` on the kernel command line.

Alternatively the max C-state can be adjusted at run time by opening the `/dev/cpu_dma_latency` file and writing a lower value (in microseconds) to the file. A value less than 10 will limit to C1, with lower values improving CPU frequency stability (down to 0). Note that the limit will be removed when the file is closed.

A simple C program to handle the `/dev/cpu_dma_latency` file is “cpudmalatency.c” and can be found in the references of this article. Note that the program must be run as root, to get write access to the `/dev/cpu_dma_latency` file.

Monitor Temperature & Limit CPU Frequency If Needed

It is a good idea to monitor CPU temperatures with something like “`watch -n 1 sensors`” if you

are planning on using the DAW for extended lengths of time at low latency settings. The CPU governor is set to performance, which drives the CPU at its maximum clock frequency. This is needed to keep up with the real-time audio work, but it also leads to higher power consumption (and heating) when DSP load is added. Critical level temperatures will cause CPU throttling and xruns in the audio server.

If temperatures reach critical levels, it is a good idea to inspect and clean the cooling system. The maximum CPU clock frequency can also be lowered to make the performance governor run the machine at a lower limit. This reduces the temperature of the CPU and can provide a more stable clock frequency, which in turn provides long session stability.

The maximum clock frequency of the performance governor can be changed by echoing new values into `/sys/devices/system/cpu/cpu{0,1,2,etc}/cpufreq/scaling_max_freq`. E.g. A machine defaults to a value of 2800000 (in kHz, which is 2.8 GHz) maximum. Echoing a value of 1800000 for example will set the new limit to 1.8Ghz, which reduces system temperature and increases stability for long sessions.

Utilize 'toram'

The Slackware Live DAW can be run at very low latency settings while operating from the USB drive. However, operations that read/write to the USB drive can cause xruns (e.g. recording audio to the USB drive, or opening programs which have to be read from the USB drive to memory). If you have enough RAM (8GB or more), the USB can be copied to RAM by booting with the parameter `'toram'`, which means that the USB drive is not read/written afterwards (unless you remount it of course).

Write to SSD

Recorded audio has to be written somewhere. Having a fast storage device like an SSD will ensure that disk writes don't bottleneck the system at the storage device.

Kill Pulseaudio

The default setup from boot of Slackware Live DAW has pulseaudio running and available to the JACK server. Having pulseaudio enabled is useful for getting audio out of programs that connect to pulseaudio instead of JACK (like a browser). However, if you don't need it, it can be stopped and will reduce the DSP load a little. E.g. On the author's machine, the JACK server idles at no-load around 3%-5% DSP when pulseaudio is running. Killing pulseaudio drops the DSP load down to between 0.1% - 0.2%

Pulseaudio is designed to auto-spawn when you log in and will re-spawn if it dies. To disable it you can disable auto-spawning in your user profile by creating or editing the file `/home/live/.config/pulse/client.conf` and adding “`autospawn=no`”. Then you can kill the pulseaudio process and it will not restart. If this change is made on a persistent USB, then subsequent reboots will keep pulseaudio disabled.

Close Extraneous Programs

Running additional programs (e.g. a web-browser) takes away from your available headroom for real-time audio work. Close what you don't need to give the minimal CPU load and more headroom.

Example Setup

Author's Setup:
Laptop: Dell XPS13 (9350), Focusrite Scarlett 18i8 (2nd Gen) USB, JACK @ 48000/48/2/Sync, toram, writing to SSD, and limited to C1. This setup can run long term without xruns, while recording multiple tracks at once. CPU temperature averages around 80C under load, but could have the frequency max limit reduced to run cooler, while still maintaining stability. Round trip latency (including hardware on USB) measures at 5.13 ms, which is good enough for live performing.

References

https://alien.slackbook.org/blog/configuring-slackware-for-use-as-a-daw/
https://wiki.linuxaudio.org/wiki/list_of_jack_frame_period_settings_ideal_for_usb_interface
https://github.com/jackaudio/jackaudio.github.com/wiki/Differences-between-jack1-and-jack2
https://github.com/Gimmeapill/xruncounter (xruncounter program)
https://linuxmusicians.com/viewtopic.php?f=27&t=19268 (xruncounter information)
https://linuxmusicians.com/viewtopic.php?f=27&t=19858&start=15 (some info on C-state performance with audio work)
https://gist.github.com/SaveTheRbtz/f5e8d1ca7b55b6a7897b (cpudmalatency program)

Sources

- Originally written by Bob Funk

howtos, multimedia, daw, audio, author 0xbf