Configuring your system for real-time low latency audio processing

Jeremy JONGEPIER
ICTE department Faculty of Humanities
University of Amsterdam
Spuistraat 134
1012 VB Amsterdam
The Netherlands
jeremy@autostatic.com

Abstract

Just installing GNU/Linux and a real-time kernel doesn't turn your system in a real-time low latency environment at once. A properly configured system demands some more modifications to what most distributions set up by default. And in a lot of cases a real-time kernel isn't even necessary. So besides going through some of the most important ways to improve the performance of your system this workshop may also debunk some tenacious myths.

Keywords

Real-time, low latency, audio processing

1 Introduction

Optimizing your GNU/Linux system for real-time low latency audio processing can be done on several levels, from uninstalling or disabling system services to modifying kernel parameters.

2 The basis

Even though the focus of this workshop is not on properly setting up a hardware rig there are a few pitfalls you have to be aware of.

2.1 Hardware

Of course there is the ever important question: is my hardware supported by GNU/Linux? Most basic hardware is well supported (CPU, ethernet, USB, FireWire, harddrive) but other hardware is either not fully supported (like some external soundcards) or can be problematic (like WiFi adapters or GPU's). Therefore it is always wise when building your audio machine to check beforehand if all components are supported. Same goes for acquiring an internal or external soundcard. You could also ask yourself the question, do I really need WiFi and a gamer GPU on my audio machine? As for the GPU, it is advisable to use a GPU that is supported by open source drivers (radeon, nouveau) so if you run into any performance issues you can file a bugreport that probably gets closed quicker than bugreports for their closed source counterparts. Besides, authorative linux audio developers have claimed the interrupt handlers of both the closed source nvidia and fglrx drivers are 'evil', i.e. you will most likely run into performance issues faster than with the nouveau and radeon drivers.

2.2 PCI(e), USB1/2/3 or FireWire?

When looking for a decent soundcard for your audio machine you have to find out for yourself what you exactly want to do with it in order to find a proper match. Keep in mind also then that there are actually four types of soundcards to choose from:

Type	Features
FireWire	High bandwidth, external, low latency, synchronous data streams, significant number of supported devices (FFADO)

USB1.1	Low bandwidth, external, asynchronous data streams, audio device class specification (any class compliant USB1.1. Audio device should work on a GNU/Linux system with the ALSA driver backend installed), large number of supported devices (ALSA)
USB2.0	Medium bandwidth, external, asynchronous data streams, no audio device class specification (so no class compliant devices which has resulted in very poor support for USB2 audio devices in GNU/Linux), small number of supported devices (ALSA)
PCI(e)	High bandwidth, internal, large number of supported devices (ALSA)

Table 1: Soundcard types

So this leaves FireWire, USB1 and PCI(e) devices as viable options. If you just need one or two inputs for your recordings and you're fine with a maximum of 2 stereo outputs then USB1.1 could be the best option. If you need more I/O then you will have to resort to FireWire or PCI(e). For mobile set-ups FireWire is very suitable, for desktops PCI(e) could be considered too. One word on USB2 devices though, there are a few supported devices available so if you really need more I/O than USB1.1 can offer and you can't or don't want to use FireWire the following devices are supported at the time of writing:

- M-Audio Fast Track Ultra
- M-Audio Fast Track Ultra 8R
- Roland/Cakewalk UA-101
- Roland/Edirol UA-1000

2.3 Which Distribution?

Basically every distribution can be used to set up a real-time low latency audio environment. But some distributions are better fitted because of the availibility of specialized audio packages, a community interested in multimedia production or simply because they're specially tailored for musicians.

Distributions that offer specialized audio packages or that have access to audio repositories are for example Ubuntu Studio (either via the official repositories or via PPA's), Fedora (PlanetCCRMA), Arch, Gentoo, Debian and Ubuntu derivates like KX Studio and Tango Studio. Ubuntu Studio also has an active community together with multimedia centred distro's like AV Linux and 64 Studio that aim specifically at musicians.

2.4 Desktop Environments, Window Managers?

Complex Desktop Environements like Gnome and KDE consume a lot of resources. On powerful workstations this shouldn't necessarily be an issue but on less powerful machines like notebooks or more specifically netbooks using the aforementioned DE's can be a showstopper. Especially when using 3D compositing Window Managers like Compiz within these DE's.

So a lot of musicians that work with mobile set-ups choose lighter Desktop Environments like XFCE, LXDE or super-light environments like OpenBox, FluxBox and IceWM. These lighter DE's mostly come with lighter WM's also, in some cases these lighter environments are actually mere Window Managers.

Since Linux Audio applications tend to be mostly modular you can easily end up with a lot of opened applications and windows and especially with smallers screens this could become cumbersome. That's where tiling Window Managers like xmonad or awesome could become handy instead of the more commonly used compositing Window Managers like Metacity (Gnome), KWin (KDE) or the 3D compositing Window Manager Compiz.

In a critical real-time low-latency audio environment it is probably the wisest option to opt for a Desktop Environment and/or a Window Manager that are as light as possible, particularly when using a mobile set-up with a notebook or netbook. Generally it is considered bad practice to

use 3D compositing and even the use of the KDE Desktop Environment is mostly avoided.

3 System Services

There are some system services that might cause xruns when they are left enabled. Some examples are for example Gnome NetworkManager, apt-xapian-index and CPU frequency scaling. The latter is not a real big issue anymore, there is now even a frequency scaling daemon available (jackfreqd) that scales the CPU frequency according to the DSP load instead of the CPU load. But if you want to be totally sure CPU scaling will not interfere with your working environment than you can always set the CPU scaling governor to performance.

Gnome NetworkManager keeps scanning in the background to check if there are any new wireless networks available. And this background scanning can cause xruns. Unfortunately Networkmanager cannot be configured to disable background scanning so if you need WiFi it is best to use another wireless network manager or to just use wpa_supplicant.

On Debian systems that have Synaptic installed the apt-xapian-index cronjob to build or rebuild the Quick Search index can be a source of xruns too, especially on less powerful machines. Best is to uninstall this package. This will disable the Quick Search field in Synaptic.

4 The kernel

4.1 Kernel parameters

On a kernel level some modifications could make a difference. First there is the default swap behaviour ('swappiness') of many distributions which is set too high. So if you use swap the system will start swapping way too quickly which might decrease overall performance. To tame this swap behaviour you could adjust the kernel swappiness parameter in the sysctl.conf file and set it to a lower value. Default is 60, 10 is recommended. While you're at it you could also adjust the max_user_watches parameter belonging to the inotify subsystem (not sure how this relates to system performance, needs some more research).

4.2 Real-time vs preemptive vs stock kernels

A stock kernel isn't optimized for real-time low-latency use. Most of the time such a kernel suffices but in situations where lower latencies are desired (MIDI, overdubbing, live on stage) using an unoptimized kernel could be the bottleneck when xruns occur at lower latencies (=< 10ms). In such cases one might have to resort to either preemptive kernels or even real-time kernels. Preemptive kernels, sometimes referred to as low latency kernels, are kernels built from upstream, unpatched kernel sources but configured optimally for use in environments where lower latencies are desired.

In some cases preemptive or low latency kernels still don't manage to provide an xrun free system. This is mainly the case when soundcards are connected to a bus that shares its IRQ with other peripherals or when extremely low latencies are needed (=< 4ms). More on shared IRQ's later. The latter case speaks for itself. This is where real-time kernels come into play, stock kernels that are patched with the real-time patchset. This patchset contains optimizations in order for a patched kernel to meet critical deadlines in environments where this is necessary, like a real-time low-latency audio environment where a MIDI note played live on stage has to be processed by the system as fast as possible so latency between the actual keypress and the outgoing sound is as low as possible.

4.3 Unloading and unbinding unnecessary drivers

The Linux kernel is monolithic and comes with a lot of drivers that are either part of the kernel or that are compiled as separate modules. Most of the time you won't be needing a lot of drivers that get loaded by default within your Linux audio environment. Drivers that are part of the kernel can be unbound from the device they're bound to and separate kernel modules can be either blacklisted so they won't load at all at boot time or they can be unloaded at any time later on.

4.3.1 Unbinding

Unbinding drivers can be useful if you want to be absolutely sure your USB MIDI controller is connected to an USB port that doesn't share its

IRQ with anything else for example. This applies specifically to stock or preemptive kernels (< 2.6.39) that won't allow you to prioritize IRQ's since they lack threaded IRQ handling. An example of unbinding an USB port of its driver:

```
echo –n "device ID" > /sys/bus/pci/drivers/ehci_hcd/unbind
```

This will disable the USB port with the given device ID. You can find the device ID of the device you'd like to unbind in the same directory as the unbind file.

4.3.2 Blacklisting

Blacklisting driver modules means you put modules on a blacklist so they won't load at boot time. This can be done in the */etc/modprobe.d/* directory on most distributions. You can either create a new file or use an existing blacklist file. If you use an existing *blacklist.conf* file on a Debian like system blacklisting a driver module is as easy as adding as adding a line like:

```
blacklist module_name
```

This will prevent the module `module_name` getting loaded at boot time.

4.4 Building your own kernel

Another possibility to get more out of your kernel could be to build your own kernel. But since the starting point of this workshop is package based distributions I won't elaborate on this subject.

5 Real-time Priorities

Linux allows to prioritize processes, including interrupt handlers.

5.1 Setting real-time priorities

On modern Linux distributions real-time priorities can be set through the PAM (Pluggable Authentication Modules) framework. By allowing a system group to use memory locking and setting real-time priorities any user that is part of that group can prioritize processes that run in userspace and any application running in userspace that needs memory locking is allowed to do so. In most of the cases an `audio` system group is used for this purpose. The following PAM options are normally added to the */etc/security/limits.conf* or */etc/security/limits.d/audio.conf* files:

```
@audio   -  rtprio     90
@audio   -  memlock    unlimited
```

These are just example settings, Ubuntu sets rtprio to 99 for example but others prefer to leave the 90-100 range for the processes that really need it and that don't run in userspace, like real-time clocks (*/dev/rtc*, */dev/hpet*). Also memory locking doesn't have to be set to unlimited because it might create a slim change some rogue process locks up all available memory. So some set it to 70% or 80% of the available RAM they have. This could result though in applications misbehaving, notably LMMS and Ardour.

5.2 Processes using real-time priorities

There are quite some applications, daemons and services that can be run with real-time priorities, as set buy either the SCHED_FIFO or SCHED_RR scheduler policy. A good example is the JACK sound daemon of which the priority can be set with the -P command line option. Examples of applications are PHASEX, Qsynth and Bristol. Applications that do not have options for running with real-time priorities can nevertheless be prioritized with the `chrt` command.

```
chrt –f –p [1..99] {pid}
```

All this allows for finegrained control over which process prevails over other processes regarding execution ie. being runnable. More info on real-time policies of the kernel scheduler, see `man sched_setscheduler`.

5.3 Controlling the priorities of interrupt handlers with rtirq

On some systems, especially notebooks, it can happen that an important controller like a FireWire controller shares its IRQ with another peripheral and that the BIOS doesn't offer the possibility to change the order of the assigned IRQ's (more info on this) or to disable any peripherals that share the same IRQ. In these cases resorting to a real-time kernel might offer a solution. A real-time kernel (or kernels => 2.6.39) comes with a task

scheduler/tasklet daemon (sirq-tasklet) that allows for prioritizing the bottom halves of interrupt handlers. The rtirq script simplifies this task by offering a single configuration file in which the affected peripherals can be listed. The rtirq script can even be used to give a higher priority to the tasklet daemon itself.

6 Disk I/O

6.1 Filesystems

What filesystem to use? In a Linux audio environment a filesystem that favours few big files over many small files and low-latency over long-term thoughput is preferable. There seems to be a consensus on using either XFS, Ext3 or Ext4. Some people favor XFS because it works well with large files, but most people are good with what their distribution uses by default which is mostly Ext3 or Ext4. Using encryption is not recommended as it may effect the amount of signal processing the system can handle. And for Ext filesystems the use noatime filesystem parameter is generally advised as it reduces the amount of disk I/O (the inode access times are not updated each time a file is read) which could improve the overall performance of your system.

6.2 Hard disk tuning

I admit, this is unchartered territory for me so if someone could enlighten us, please go ahead!

7 The Jack Audio Connection Kit

The indispensible tool for GNU/Linux to work with audio on a professional level.

8 Using FireWire

Access to FireWire devices.

9 Conclusion

Concluding text

10 Acknowledgements

Our thanks go to …

References

[1] M. Kay. 1986. Parsing in Functional Unification Grammar. In K. Spark Jones B. J. Grosz and B. L. Webber, editors, *Readings in Natural Language Processing*, pages 125-138. Morgan Kaufmann Publishers, Los Altos, California.

[2] Frederick Mosteller and David Wallace. 1964. *Inference and Disputed Authorship: The Federalist.* Addison-Wesley, Reading, Massachusetts.