Multimedia Support in the Linux Kernel

KERNEL SOUNDS

We'll show you how to tune up your Linux system for multimedia applications. **BY DAVE PHILLIPS**

In the late 1960s, "multimedia" was a new term connected typically to the work of artists such as Andy Warhol. Although Warhol was not the only artist working in this domain, his Exploding Plastic Inevitable performances defined the multimedia event. An EPI show simultaneously included film, projected images, dancers, music, recitations, and so on, for an effect calculated to overwhelm the senses. By comparison, the multimedia experience on your computer is relatively restrained, but it can be colorful and exciting.

A computer-based multimedia production comprises text, graphics, sound, and video, but these elements can be combined into a media-rich presentation, an interactive audio/video installation, or a stroke-inducing first-person action game. Each combination is a true multimedia production. Like the original multimedia events, they require abundant resources. Modern machines have plenty of drive space and RAM, fast CPUs, and powerful video capabilities, but modern multimedia software can test the performance envelope of even the most powerful desktop computer.

In this article, I look at the Linux kernel's integral support for the hardware and software required by machines intended for the production and presentation of rich media formats. Because of space limitations, I can only touch on some of the factors required to optimize that support, but I hope my efforts inspire you to compile your own media-optimized kernel. The source code is free, the build process isn't terribly complicated, and the results can yield a considerable improvement in audio and video performance.

Configuring the Kernel for Multimedia Machines

The Linux kernel provides support for multimedia with drivers for a variety of devices (sound cards, video boards, graphics tablets, etc.) and code for features of various motherboard chipsets. The kernel's openness and modularity invite extension by developers who want to add new capabilities to the existing kernel services.

Before you compile the kernel, you must configure its options, usually with the help of a menu-based utility that presents the configuration options and preferences in an organized GUI (Figure 1). The scope of this article is restricted to those parts of the kernel configuration that apply to the topic (i.e., support for multimedia), and the following descriptions and explanations assume some experience in compiling programs from source code. However, even if you're a complete novice, Google can direct you to text and video guides for the process. It's not terribly difficult, and if you're patient and thorough, it might be fun.

CPU Type

The first settings will be made in the *Processor type and features* section. The processor type is an easy choice, but if you're not sure what CPU is in your machine, just issue the *uname -a* command at a terminal prompt.

This tool will respond by listing various facts about your hardware and its operating system. For example, on my notebook, I'm running Ubuntu 8.10 with a kernel patched for real-time operation. The uname utility reports the following information about my system:

```
dlphilp@maximus:~$ uname -a
Linux maximus 2.6.27-3-rt #1 ⮐
PREEMPT RT Mon Oct 27 ⮐
03:05:19 UTC 2008 i686 GNU/Linux
```

From this report, I learn that my kernel is numbered 2.6.27-3-rt, that it's running on the first (*#1*) of a dual-core machine, and that it has been compiled for full preemption. Also, I know that my CPU type is an i686, a post-Pentium processor type. Actually, it's an AMD Turion-X2, configured as a single-core 32-bit i686 (with the *CONFIG_X86* kernel option) so that I can use certain software that will not run on a dual-core 64-bit machine.

Kernel Timers

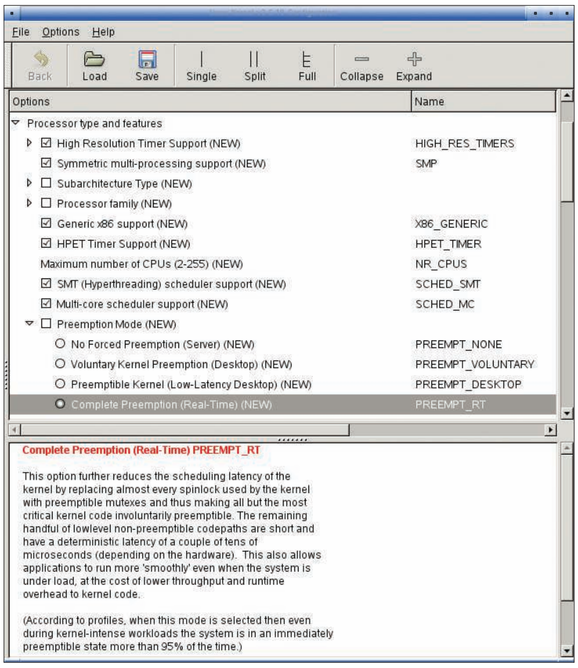
Successful multimedia performance of any kind is critically dependent on timing. Audio and video need tight synchronization, and sound needs to be free of dropouts and spurious noise. Fortunately, the modern Linux kernel provides the necessary components, but your distribution might not have the options enabled for those timers. To acquire the benefits of better timing, you might need to recompile your kernel.

The high-resolution timers option (*CONFIG_HIGH_RES_TIMERS*) enables a "tickless" system with a timing accuracy of about 1msec on most contemporary machines – a considerable improvement over the standard hertz-based timer resolution. Introduced in kernel 2.6.21, the tickless system reduces the load on the system clock by shutting off the timer interrupt (the tick) whenever the system idles. This process saves power on laptops and notebooks and improves multitasking performance. The option for high-resolution timers can be enabled without the tickless system, but in an optimal multimedia system, make sure that *CONFIG_NO_HZ* is set to true.

The high-performance event timer (HPET) was once known as the multimedia timer. According to its entry in Wikipedia, this timer "… can produce periodic interrupts at a much higher resolution than the RTC [real-time clock] and is often used to synchronize multimedia streams, providing smooth playback and reducing the need to use other timestamp calculations." The option (*CONFIG_HPET_TIMER*) is machine dependent and will not work on older hardware or operating systems, including the Linux 2.4 series. It also requires the rtc-cmos driver instead of the traditional RTC driver discussed next.

Which of these timers should you use? On my JAD 1.0 box (openSUSE 10.2), the HPET and the high-resolution timer options are both compiled into the kernel. Clocks and timers operate transparently on Linux, so don't worry about loading modules or installing other control software. The kernel module loader handles everything. You did enable the support for loadable modules, didn't you?

At this point, I'll include a word about Ye Olde Way of configuring high-resolution timing at the kernel level. Users of pre-2.6 kernels might not have access to the new timers, but they can still set up their kernels for enhanced resolution. The *CONFIG_HZ* option allows frequency settings for 100, 250, and 1,000Hz, the last of which is the preferred resolution for any system running audio and MIDI applications. Conventional wisdom advises compiling the RTC driver, again a necessity for high-resolution audio and MIDI event timing. Note that the timer frequency option is available in the *Processor type and features* section of your kernel configuration, but the RTC driver is selected in the *Device Drivers* section.



Options	Name
Processor type and features	
☑ High Resolution Timer Support (NEW)	HIGH_RES_TIMERS
☑ Symmetric multi-processing support (NEW)	SMP
☐ Subarchitecture Type (NEW)	
☐ Processor family (NEW)	
☑ Generic x86 support (NEW)	X86_GENERIC
☑ HPET Timer Support (NEW)	HPET_TIMER
Maximum number of CPUs (2-255) (NEW)	NR_CPUS
☑ SMT (Hyperthreading) scheduler support (NEW)	SCHED_SMT
☑ Multi-core scheduler support (NEW)	SCHED_MC
☐ Preemption Mode (NEW)	
○ No Forced Preemption (Server) (NEW)	PREEMPT_NONE
○ Voluntary Kernel Preemption (Desktop) (NEW)	PREEMPT_VOLUNTARY
○ Preemptible Kernel (Low-Latency Desktop) (NEW)	PREEMPT_DESKTOP
● Complete Preemption (Real-Time) (NEW)	PREEMPT_RT

Complete Preemption (Real-Time) PREEMPT_RT

This option further reduces the scheduling latency of the kernel by replacing almost every spinlock used by the kernel with preemptible mutexes and thus making all but the most critical kernel code involuntarily preemptible. The remaining handful of lowlevel non-preemptible codepaths are short and have a deterministic latency of a couple of tens of microseconds (depending on the hardware). This also allows applications to run more 'smoothly' even when the system is under load, at the cost of lower throughput and runtime overhead to kernel code.

(According to profiles, when this mode is selected then even during kernel-intense workloads the system is in an immediately preemptible state more than 95% of the time.)

Figure 1: 64 Studio's kernel configuration GUI.

Kernel Preemption

Timers aren't the only items of interest in this section; you can also decide on the degree of preemption you intend for your kernel, but first, consider the topic of preemption in the Linux kernel.

As users' needs have become more sophisticated, their demands on computer hardware have become greater. Multitasking is an ordinary aspect of modern computing, and although multitasking is indeed a lovely thing, it also creates a world of concerns for multimedia production and playback. Without some sort of process control, applications will clash over access to system resources and services, causing audio dropouts, stuttering video, and even complete system lock-ups. Under normal circumstances, most users will be happy with a non-preemptive kernel, but if your applications need priority access to resources and services, the normal kernel scheduler might not suffice. Preemption is a way to guarantee an application's priority status, keeping other processes from interfering with its operations. For example, when I record with Ardour, I need to be absolutely certain that no other process is going to knock Ardour aside. The preemptive kernel saves the day.

Preemption is available in four modes. The first sets no preemption at all and is the default for servers. Voluntary preemption, the second mode, is the default for normal desktop use without timing-critical applications. The third option provides a low-latency system for users who want an optimal system for multimedia performance from applications such as games and audio/video players. The fourth choice, complete preemption, is for users who want a media production platform, either in a studio setting or as a real-time performance system. Your choice of preemption mode depends on your needs, so choose wisely. As you shall see, a preemptive kernel can have its own problems.

Device Drivers

Without the correct drivers, your shiny new cards and USB add-ons won't yield so much as a squawk, so again be sure you've selected the right drivers for your hardware. Although you can build all the available drivers as modules for the kernel to load as needed, if you must be conscientious about space, you can build only the modules you need for your current hardware. For a multimedia system, look closely at kernel support for your graphics hardware, video devices, and sound cards.

Kernel drivers for graphics adapters have code for hardware from various manufacturers. Just find your device and select the proper driver. However, bear in mind that these drivers work closely with the X Window System, and the overall performance of your graphics display(s) will be determined by the optimal combination of X and the kernel driver. Other potentially relevant options include */dev/agpgart* support for older machines, a direct rendering management system for certain video chipsets, and support for framebuffer devices.

The default kernel sound system is from the Advanced Linux Sound Architecture project (ALSA). To activate sound support in the kernel, select the *CONFIG_SOUND* and *CONFIG_SND* options. Next, choose the driver corresponding to your sound hardware. This step is potentially confusing: Your sound card might not be listed, but its chipset could be supported by the ALSA drivers. For a complete list of supported devices tabulated by sound card, manufacturer, and chipset names, see the ALSA website's SoundCard Matrix.

If your sound hardware is connected to a USB port, you'll need to enable the *CONFIG_SND_USB* and *CONFIG_SND_USB_AUDIO* options. PCMCIA cards will require the *CONFIG_SND_PCMCIA* option, and similar options exist for sound support with PowerPC, Sparc, ARM, and other non-x86 Linux-capable platforms.

The kernel configuration utility lists Open Sound System (OSS) modules as deprecated, but they are still usable. The older OSS system can be employed if ALSA can't be used, but I would advise the use of the OSS/Free package [1] instead of the kernel's OSS modules. OSS/Free is now open source software, and the current package is an exceptionally stable and powerful alternative to ALSA. Also, it is easy to install and can be used within an otherwise ALSA-based system.

Multimedia Devices

The *Multimedia Devices* section includes essential support for video and radio capture devices. If you work with a webcam, a television card, or a radio tuner, you'll need to activate the video4linux (V4L) driver by selecting the *CONFIG_VIDEO_DEV* option. Recent kernels include extensive support for a variety of video capture devices, so be sure to scan the list of supported hardware. First select the relevant option(s), and be sure to enable the *CONFIG_V4L_USB_DRIVERS* option if you plan to use a USB-connected capture or encoding device.

As previously mentioned, the drivers section includes the option to build the driver for the enhanced real-time clock (*CONFIG_SND_RTCTIMER*). When this option is selected, you will be given the opportunity to declare RTC as the default timer for the ALSA sequencer (a MIDI dataflow manager). Consider this option a necessity. This section also includes an option for building the driver for an IEEE 1394 port, more familiarly known as the Firewire port. Firewire is an excellent data transfer protocol, designed to handle high-capacity streams, but Linux support is relatively new. This option includes a stable and an experimental driver; choose the option that best applies to your hardware.

Disk drive performance should be optimal. The drive type (IDE, SCSI) is not so important as it once was, but if your system includes IDE drives, you should check a few options in the subsection for *ATA/ATAPI/MFM/RLL* support. In the *IDE/ATA/ATAPI Block Devices* section, you need to find and accept the options for *Generic PCI Bus-master DMA Support* and *Use PCI DMA By Default When Available*. These options enable DMA (direct memory access) for your disks, which is helpful with low-latency disk-intensive work. However, even with these options, you might still need to optimize your disk performance with the hdparm utility. The following command enables 32-bit I/O and turns on DMA support for the IDE hard drive in my JAD 1.0 machine:

```
hdparm -c 1 -d 1 /dev/hda
```

For a status report on the indicated device, run the utility without parameters. Incidentally, if you know nothing about hdparm, its manual page is required reading before you use the tool. Some of hdparm's parameters are dangerous, but fortunately, you need only the two flags shown above to improve the performance of your IDE drives.

Miscellaneous Settings

The *Block Layer* configuration section includes three options for choosing your system's I/O scheduler. The *Anticipatory scheduler* is the normal system default, the *Deadline scheduler* is recommended for systems running large databases, and the *CFQ scheduler* is advised for load averaging on normal desktop systems. If you choose to build all three schedulers, you must declare one to be the system default. The CFQ option is the scheduler of choice for low-latency systems.

The *Kernel Hacking* section includes support for the Magic SysRq key (*CONFIG_MAGIC_SYSRQ*), a handy amenity for anyone using a preemptive kernel. This option enables the use of the SysRq key in combination with other keys to recover a frozen machine or reboot a machine without corrupting the filesystem. The Magic SysRq Key page on Wikipedia has a full list of magic key combinations.

Latency and the Real-Time Kernel

On the Wikipedia disambiguation page for "latency," numerous references share the common identifying factor of a time delay between an event's initiation and its realization. Under normal circumstances, latency might not be an issue, but as playback and production demands increase, so does the possibility for disruptive latency. Much of the information here is directed toward lowering latency, but a normal Linux kernel cannot reach the range of latency considered to be acceptable in professional applications. Fortunately, thanks primarily to the work of kernel developer Ingo Molnar, a set of patches are available that can dramatically reduce latency in the kernel – down to and beyond professional limits [2]. Most of the media-optimized Linux distributions include a real-time kernel, and most of those kernels are patched with Molnar's work.

Table 1: Machine Details at Studio Dave

Machine	CPU	Distribution	Kernel	nVidia Driver
Big Black	AMD 3200+ (64-bit)	64 Studio 2.1 (Debian)	2.6.21-1	169.12
The3800	AMD 3800+ (32-bit)	JAD 1.0 (openSUSE)	2.6.19-5	169.12
Maximus	AMD Turion-X2 (32-bit)	Ubuntu 8.10 (Debian)	2.6.27-3	177.80

Molnar has identified six traditional sources of latency in the Linux kernel:

- Calls to the disk buffer cache
- Memory page management
- Calls to the */proc* file system
- VGA and console management
- Forking and exiting large processes
- The keyboard driver

These factors can delay the return of control to the scheduler for several milliseconds, and with enough delay, audio dropouts and video frame loss will occur. Molnar's patches tune these factors until kernel latency reduces to less than 5msec, a very dramatic reduction from the latency of an unpatched kernel and within professional limits. However, low latency also depends on audio hardware capabilities, and in some instances, latency might not be so reducible.

The term "real-time," with regard to the kernel, has two rather different meanings. The real-time kernel I've been configuring here is "soft"; that is, it can promise to deliver the goods within a certain time, but it can't make a guarantee. By contrast, a hard real-time system guarantees the delivery within a specified time frame. In neither of these usages do you find any necessary reference to low latency, although typically, a real-time system operates within tight time constraints, even down to microsecond levels. The essential difference between hard and soft real-time systems is criticality: If Ardour suffers an audio buffer overrun, I might get upset over the resulting glitch in my recording, but if the real-time control system software for a nuclear power plant misses a few beats, then we're all in serious trouble.

The Real-Time Kernel in Real Life

Table 1 compares the three machines I use here at Studio Dave. All systems are configured for *PREEMPT* and *RT*, and all machines have nVidia graphics chipsets. Big Black is my main production box, which I use for recording with Ardour, composing with Csound5, producing CD/DVDs, and managing all my teaching-related activities. The3800 is used mostly for composing with MIDI, watching movies and DVDs, and running Windows music and sound software under Wine. Until recently, it has also been my box of choice for surfing the video web, but that could change with the availability of 64-bit Flash and the 64-bit Java plugin. Maximus is my cutting-edge machine; it's used mostly for testing software that requires the latest Qt and Gtk, such as Qtractor and Ardour3.

Big Black and The3800 run on distributions optimized specifically for enhanced multimedia performance. Both machines include M-Audio Delta 66 audio interfaces, and both report 5.8msec latency in the JACK audio server. Maximus runs on a Molnar-patched real-time kernel; otherwise, its system is unmodified Ubuntu. It is also the least stable of the three systems, and I don't plan to use it for audio or video production purposes. Maximus is a notebook with an Intel HDA audio chipset, but I've added an Edirol UA25 USB interface to its audio capabilities. Alas, I have yet to bring latency below 11msec without xruns (ALSA-speak for buffer excess or insufficiencies), but I still have some module options to explore.

All of my machines are set up to run Jean-Pierre Lemoine's AVSynthesis, a program that manipulates and blends image and sound into fantastic animations. Because it needs abundant resources, it's an excellent application to use to test multimedia capabilities. Its sound production relies on Csound5 compiled for high-definition audio, its video capabilities depend on OpenGL and hardware-accelerated 3D graphics, and it can be operated in real time.

I also looked for information regarding kernels used by other optimized distributions. Fedora-based Planet CCRMA [3] offers stable real-time kernels from the 2.6.24 series and testing kernels from the 2.6.26 releases. Debian-derived Musix [4] uses a stable 2.6.21 real-time kernel in its 1.0 R2 release. Gentoo-based Bardix 0.1 [5] employs a real-time 2.6.25 kernel. From this information, you can see that distributions optimized for multimedia production and performance clearly prefer the 2.6 kernel series.

Significant Non-Kernel Factors

When considering your hardware, you should exercise wisdom. Bigger and faster are the watchwords for a powerful multimedia system, but some caveats remain. Video is an especially tricky factor. If, like myself, you need to use nVidia's closed source drivers, you must resign yourself to the possibility of inexplicable problems when running a real-time kernel. Because of its closed source nature, you can't know how the video driver interacts with the kernel at levels that affect latency and real-time response. To nVidia's credit, they do try to keep up with Linux kernel development, but the real-time patches are not official patches, and until nVidia opens their source code, kernel developers are unable to help users who experience performance problems on real-time and low-latency systems.

Also, you should keep your X windowing system current. X works closely with the kernel video drivers, so to ensure maximum performance, make sure that X is up to date.

Outro

This topic goes much deeper than I've been able to discuss here, but the web is rich in relevant resources. For example, you should have no trouble finding out more about kernel configuration [6], low latency [7], and real-time optimizations [8], and I hope that some of you will take the next steps toward compiling your own kernel. Rolling your own is a time-honored Linux tradition, and you don't need an engineering degree to do it. Just be sure to keep your old kernel around for booting into in case things go wrong, breathe slowly and deeply, be patient and brave, ask questions, and, above all, have fun. ■

INFO

[1] OSS/Free: *http://www.4front-tech.com/usslite/*

[2] Ingo Molnar's real-time kernel patches: *http://www.kernel.org/pub/linux/kernel/projects/rt/*

[3] Planet CCRMA: *http://www-ccrma.stanford.edu/planetccrma/software/*

[4] Musix: *http://www.musix.org.ar/en/index.html*

[5] Bardix: *http://www.linuxmusicians.com/viewtopic.php?f=4&t=696??*

[6] LinuxForums' guide to compiling the Linux kernel: *http://www.linuxforums.org/forum/linux-kernel/55612-mini-howto-compile-linux-kernel-2-6-a.html*

[7] The low-latency HOWTO: *http://lowlatency.linuxaudio.org/*

[8] The Real-Time Linux wiki: *http://rt.wiki.kernel.org/index.php/Main_Page*