Serving the PC enthusiast for over $5x10^{-2}$ centuries

by Charles "ctkrohn" Krohn

12/24/2003 Edition

Welcome to this week's edition of Linux.Ars. Today we feature a detailed description of one of the most important parts of the newly-released Linux 2.6

kernel: the scheduler. The new scheduler features several improvements over that in 2.4; we will not only explain the improvements, but also describe how the scheduler works and why these improvements are important. The excellent media player Zinf is our Cool App of the Week: its music browser offers an alternative to the iTunes method of organizing one's music.

Inside the Linux 2.6 Scheduler

At long last, the 2.6 kernel is finished. The Christmas holiday gives us plenty of free time to play with its new features. The list of changes is quite long; very much work has been done on this new kernel. One of the most important and visible changes it the introduction of the O(1) scheduler. The scheduler is the piece of the kernel which allocates slices of time to individual programs running on the computer. It makes it possible for one CPU to execute multiple programs at once by allowing one program to run for a certain amount of time, then switching to another program, allowing it to run for another amount of time, and so forth. As such, a good scheduler which allocates CPU time efficiently can lead to a much more responsive user experience.

What the heck is O(1) anyway?

"O(1)" is an example of "Big-Oh" notation, a notation used in computer science to describe the running time of an algorithm. An algorithms is a well-defined, step-by-step method for accomplishing a specific computational task. Big-Oh notation does not actually measure the number of milliseconds that an algorithm takes to execute; such a measure is highly dependent on the machine specifications, operating system, etc. Rather, it measures how long an algorithm takes in proportion to the size of the input. For example, the insertion sort algorithm is $O(n^2)$, meaning that the time it takes to sort a list of items is proportional to the square of the size of the input.

It is easy to recognize the input of most algorithms. For example, the input to the insertion sort algorithm is a list of items. Scheduling algorithms pick a task to execute and decide how much time to allocate it; their input is the set of tasks that the operating system is dealing with. When we say that the scheduler is O(1), we mean that it takes a constant amount of time to execute, independent of the number of tasks. Whether

Recent Articles

IBM ThinkPad T40 review

ViewSonic airpanel V150p review

Adobe InDesign CS review

Compaq Evo N620c laptop review

Mac OS X 10.3 — the *definitive* review

Dealing with PlayStation 2 disc read errors

iTMS: Facing New Challenges

Quickeys X2 for OS X

PowerPC & Intel 64-bit compiler update

GNOME 2.4 Desktop & Developer Platform

Macintosh Browser Smackdown

Best anti-spam solutions for Windows

Antec Sonata Case: a quiet wonder

Digital Video Cleaning without the Elbow Grease

Ars for Five Years

/etc

OpenForum

Distributed Computing

Take the Poll Technica

FAQ: Celeron overclocking

your computer is executing 1000 tasks simultaneously or 10, the scheduler will always take the same amount of time to pick the next task. This is a very desirable characteristic for a scheduler.

Schedulers and the processes they schedule

To understand how the scheduler allocates CPU time and manages program execution, it is necessary to understand the concept of a "process." A process is a set of instructions that the processor executes sequentially. One or more processes can make up a program.

Processes can have one of five states: running, interruptible, uninterruptible, zombie, or stopped. In this article, we will only consider the running and interruptible states. A process in the running state is either currently being executed by the processor, or stored in memory awaiting execution. An interruptible process is one that is currently "blocking." Blocking occurs when a process voluntarily yields control of the CPU until a certain condition is met. This typically occurs when a process is waiting for input from the user, hard disk, network, or other external resource.

The Linux 2.6 scheduler

by Amir Elaguizy and Jim "Spamizbad" Battin

Processes can have various priority levels. The Linux 2.6 scheduler uses an efficient algorithm to favor higher-priority processes while still allowing lower-priority processes to execute. The scheduler keeps a list of the priority levels. When it comes time to select a process, the scheduler finds the highest priority level containing available processes. It then selects a process from the desired priority level. Because the number of priority levels is fixed, the scheduler always takes a constant amount of time.

The scheduler has another important ability: preemption. Preemption enables a process to be stopped at any time, allowing a higher-priority process to be started. This is very important from a user's point of view; it allows processes which interact with the user to be executed whenever necessary, while allowing lower-priority process which do not interact with the user to run in the background. For example, imagine you are running a distributed computing client. This program uses the majority of your CPU time when you are not in front of your computer. The

minute you begin using a program such as a web browser, the web browser "preempts" the distributed computing client. Your system will not appear sluggish, even though you have a computationally-intensive process working in the background.

The benefits of preemption are great enough that the kernel developers decided to make the kernel itself preemptible. This allows a kernel task such as disk I/O to be preempted, for instance, by a keyboard event. This allows the system to be more responsive to the user's demands. The 2.6 kernel is able to efficiently manage its own tasks in addition to user processes.

Runqueues and load balancing

Linux 2.6 is able to efficiently manage more processors than Linux 2.4, the previous stable version. The ability to deal with different amounts of computational resources, from small embedded CPUs to massive 64-processor supercomputers, is called "scaling;" it is one of the new kernel's many strengths. One might think that a scheduler designed to manage one CPU could never be adapted to managing 64 CPUs; however, the Linux 2.6 scheduler can manage large multiprocessor systems through the use of special lists called "runqueues." A runqueue stores information about the processes running on a single CPU; there is one runqueue for each CPU in the system. The information contained in the runqueues allows the processor to seamlessly transfer control from one CPU to another, using a method called "load balancing."

Load balancing is a way to ensure that no CPU's resources are going to waste while another CPU is overstressed. If the scheduler finds that one runqueue has many more processes in it than another, one or more processes may be moved from the larger runqueue to the smaller one. The load balancer is invoked whenever a runqueue is empty; if no runqueues are empty it is invoked periodically. The periodic timer allows the system to maintain a reasonable load balance across many CPUs without devoting too much time to moving processes from one CPU to another. Balancing the load whenever a runqueue is empty allows the scheduler to ensure that precious CPU power never goes to waste. There is one exception to this load balancing rule — some special processes may be fixed to a certain runqueue. This attribute is called "thread affinity;" a thread is simply

another name for a process.

It is important to note that if SMP (symmetric multiprocessing) support is not enabled when one compiles the kernel, none of the load-balancing code will be enabled and no time is wasted trying to balance the load on a single CPU.

Timeslices and niceness

You may be wondering how much time the scheduler allocates to each process. The amount of CPU time which the scheduler assigns to a particular process is called a "timeslice." After a process' timeslice is used up, the process is stopped so the next process can execute. It is important to remember that a process can be stopped in the middle of its timeslice; this is the purpose of preemption. Different processes are assigned different timeslices based on priority; higher-priority processes run for longer than lower-priority ones. Priority is not a unitary concept; each process has static priority and dynamic priority.

Static priority, or "niceness" in traditional UNIX terminology, is a measure of how important a process is. It can be set by the user or by other programs. Processes with a low nice value are granted longer timeslices; those with high nice values are granted smaller ones. Nice values range from -20 to +19. The fact that high-priority processes have low nice values may seem confusing if one views niceness as a measure of priority; rather, think of niceness as a measure of how willing the processes is to yield to others. Nice values can be set on the command line with the "nice" command, or through some system monitor programs. The November 5 Linux.Ars has a good introduction to managing processes with such tools.

The dynamic priority of a process is determined by the scheduler by monitoring its behavior during execution. Processes which spend much of their time blocking are known as "I/O bound" processes; their behavior is bound by input and output. When the scheduler recognizes an I/O-bound process, it is granted a negative bonus (down to -5), and thus a larger timeslice. By contrast, CPU-bound processes are granted a positive bonus (up to +5), thus a smaller timeslice. This prevents CPU-bound processes from controlling the processor, and allows input and output to proceed smoothly. Often, a single process may switch between I/O-bound and CPU-bound behavior.

For example, a process might read some information from the keyboard, and then perform some computations based on that input. The scheduler needs to constantly be aware of a process's behavior; dynamic priorities may then be readjusted accordingly.

The scheduler keeps track of all these timeslices with two lists of processes. The first list contains processes that have yet to use their timeslice; the second contains those processes which have used their time. When a process uses its timeslice, the scheduler calculates a new timeslice by adding the dynamic priority bonus to the static priority. The process then gets inserted in the second list. When the first list becomes empty, the second list takes the place of the first, and vice-versa. This allows the scheduler to continuously calculate timeslices with minimal computational overhead.

Why must the scheduler be so careful when calculating timeslices? A good scheduler must achieve a proper balance between throughput and latency. Throughput is the amount of data that can be transferred from one location to another. Latency is the time it takes for a process to respond to input. This balance is achieved by adjusting timeslices. An I/O-bound process needs good throughput if it is to accomplish its tasks quickly. This is why the scheduler gives I/O-bound processes large timeslices; they have make and respond to I/O requests, and don't have to wait as long for other processes to execute. Because nearly all processes can benefit from superior throughput, why not give all processes a large timeslice? If a scheduler were to do this, latency would suffer. Because each process has a long time to complete its task, other processes won't be able to respond quickly to user input. A good balance between throughput and latency leads to a responsive user experience with sufficient throughput.

Conclusion

The Linux 2.6 kernel provides a powerful foundation for the newest Linux distributions, programs, and devices. An immense effort was put into the design and testing of this new release; this effort was so great that many people thought that the kernel should be numbered 3.0 rather than 2.6. Developers such as Ingo Molnar, Robert Love, Con Kolivas, David Libenzi, and Linus Torvalds, in addition to thousands of other developers and testers, have brought us an excellent release. There are many other important areas of the

kernel which have been overhauled, such as key parts of the I/O subsystem; sadly, these are beyond the scope of this article. If you are interested in kernel issues, we recommend Kerneltrap and Kernel Newbies. More ambitious users may wish to subscribe to the Linux Kernel Mailing List, the official medium for kernel development conversation.

Cool app of the week: Zinf

As people continue to acquire more and more music files, it becomes very difficult to organize them. Simple media players such as XMMS or its GTK2 port Beep have a single flat playlist. If one wants to find a certain song, there are two choices: scrolling through a list or, if feeling lucky, searching for the song by name. The first option requires that the user know exactly which songs are in which parts of the playlist; the second requires that the user type text in order to find the song. Neither solution is satisfactory for large music collections; the ideal music player would allow the user to select any song with a few clicks of the mouse.

There are many programs for all major platforms that allow users to maintain and browse large music collections; Apple's iTunes is probably the best-known. On Linux, two of the more popular ones are Juk for the KDE desktop environment and Rhythmbox for GNOME. Both these programs mimic the iTunes interface very closely.

While some users like the iTunes/Rhythmbox interface, I prefer a different screen layout. The music browser interface has, in my opinion, a few flaws. The lower pane repeats much of the information which is displayed in the upper two panes; the upper two panes also display much information which is not relevant to the current playlist. A better way to design a music browser is to use a tree view, similar to that of most file managers. Artists would be the top-level nodes in the tree. Click on the "expand" icon next to an artist's name, and a list of albums would be displayed. Expand an album, and see the songs it contains. Such a browser does not take up a great deal of horizontal space; instead, it makes good use of vertical space. If placed on the left side of the screen, like the tree views in most file managers, it could provide an efficient way to browse one's music collection while still leaving enough space to see the current playlist.

My second complaint about the iTunes/Rhythmbox interface is that it does not provide a quick and easy way to see which songs are currently queued for playing. It is possible to create a "Now Playing" playlist and add things to it on the fly; however, one must be constantly switching back and forth between the Now Playling list view and the library browser view. Because the tree view left us with extra screen real estate, we can easily put a "Now Playing" list on the right-hand side of the music browser window; that way we can browse our library and manipulate the current playlist without changing between views.

Zinf implements these two ideas effectively; its music browser (screenshot) is a joy to use. While its music browser is close to perfect in my opinion, there is more to a media player than the browser. I have two major complaints about its design. The first is that Zinf uses an annoying two-window interface: one window for controlling playback (with the traditional play controls and an equalizer), and another window for the music browser. It would be much nicer if Zinf had an integrated design like iTunes and Rhythmbox. The second complaint is that Zinf has a dated GTK1 user interface. Zinf has not seen any releases since August and the development mailing list is quiet; this makes it unlikely that it will ever be ported to GTK2. While it lacks many modern bells and whistles like integrated CD ripping and burning, Zinf is a relatively mature program and does not have the stability programs of some newer applications like Rhythmbox. In addition, it runs well on Windows.

While RPM packages do exist for RedHat 9, they do not work well in Fedora. If you wish to use Zinf in Fedora, your best bet is to compile the program from scratch. There are Debian packages in the main apt repository. Users of other distributions may be able to find packages; if not, it is an easy application to compile.

/dev/random

We're always looking for interesting ideas for articles. If you are interested in any particular facet of Linux, whether it be the new I/O subsystems in Linux 2.6 or advanced package management techniques, we want to hear about it. Post your ideas in the discussion page for this article.

Last week, we said that in this issue, "we'll explore how to take advantage of this flexible boot process in order to have the computer boot off the network, without requiring any disks at all." Unfortunately; this article must wait until a later date.

New NVIDIA drivers were released on December 22nd. Check it out. It's possible to patch these drivers for use with Linux 2.6

Support was recently merged into OpenBSD to allow two boxes to synchronize their firewall states over a physical LAN interface. If this is stable by the time 3.5 comes out, OpenBSD will be very handy for deploying redundant firewalls

Linux.Game.Ars: Bioware has announced the availability of Linux binaries for its new Neverwinter Nights expansion pack, "Hordes of the Underdark." More information is available here.

Previous editions

Revision History

Date	Version	Changes
12/24/2003	1.0	Release

Discuss this article

Back to Ars Technica