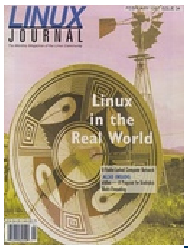
What Is Multi-Threading?

Feb 01, 1997 By Martin McCarthy (/user/800247)
in

With the 2.0 kernel, most Linux users now have the capability of using multi-threaded processes—well, what does that mean?



(/issue/34)

From Issue #34
February 1997 (/issue/34)

Perhaps one of the reasons you use Linux is because it is a multi-tasking operating system. In which case you are probably well aware of the utility of being able to do more than one thing at a time—perhaps compiling the utility which will bring you fame and fortune whilst editing a letter of resignation to your boss. So you might recognise the utility of an individual process that can do more than one thing at once.

When might this be a good idea? One application for multi-threading is a program which relies on a large number of very similar and independent mathematical operations—the oft-quoted example of this is matrix multiplication. We look at a simple example of a multi-threaded matrix multiplier later in this article.

Another type of application which can benefit from multi-threading is a server application. Whenever a client attempts to connect to the server, a new thread can be created to look after that client whilst the “watcher” thread continues to wait for more clients to connect.

“But wait!” I hear you cry. “Lots of server applications already work like that, but they simply fork another process rather than starting another thread.”

“You're right...” I reply.

“Don't interrupt,” you continue. “This sounds like another way of doing the same thing, but for no good reason other than to say it's a `multi-threaded application' and so you can bump up the price to those who like to be blinded by science.”

At this point I decide to ignore any more of your interjections and just explain.

Yes, creating a new thread is very similar to forking a new process, but there are differences. When a new process is forked, it shares relatively little data with the parent process which created it; when a new thread is created, it shares much more information (such as all the global variables and static local variables, the open files, and the process ID). The overhead of creating separate copies of everything makes the creation of a new process slower than the creation of a new thread. And the time it takes to pass control from one process to another (a *context switch*) is longer than the time required to pass control from one thread to another. Also, since the threads share their data space, passing information from one thread to another is much more straightforward than passing information from one process to another—while this does have its own problems, it need not be difficult if a little care is taken. A simple example of a multi-threaded server is given below.

A third type of application which could benefit from multi-threading is a Doom-like game where each of the computer-controlled baddies has its own “intelligence” and can act independently. The main game-play part of the program could be in its own thread (or multiple threads), there could be another thread handling each of the characters in the game who are controlled by real people, and yet more threads for each of the characters controlled by the computer.

POSIX Threads

There is a POSIX standard for multi-threading: 1003.1c. If you want to write portable programs (which doesn't seem like a bad idea to me), it would be wise to stick to this standard rather than use non-POSIX conforming libraries or using the raw system calls which Linus so kindly provided in the Linux kernel (about which I say just a little more later).

The examples in this article use POSIX multi-threading functions called from C. However, the concepts in some non-POSIX libraries and systems are often very similar. Solaris threads are easily understood by someone familiar with POSIX threads, and while Java threads and the multi-threading in the Win32 and OS/2 APIs are a little different, there should be little in these which would leave you quaking in your boots.

Linux Threading: Linus Gave Us **clone()**

Multi-threading capability is included in the version 2.0 Linux kernel (and many version 1.3 kernels). The *clone()* system call creates a new *context of execution*, or “COE” (to use Linus' term), which can be a new process, a new thread, or one of a range of possibilities which doesn't fit into either of these categories. The *fork()* system call is actually a call to clone() with particular values as parameters, and the **pthread_create()** function call could be a call to clone() with a different set of values as parameters. clone() is used by some implementations of multi-threading libraries to provide *kernel threads*, but further discussion of clone() is beyond the scope of this article.

____

Comments

Comment viewing options

Threaded list - expanded ▼ | Date - newest first ▼ | 50 comments per page ▼ | Save settings

Select your preferred way to display the comments and click "Save settings" to activate your changes.

Nice Article.. very usefull (/article/1363#comment-356233)
Submitted by Anonymous (not verified) on Fri, 09/24/2010 - 07:03.

Nice Article.. very usefull for basic understanding of multithreading.

Thanks.

What is multithreading (/article/1363#comment-348594)
Submitted by Anonymous on Thu, 02/25/2010 - 02:19.

to know more about multithreading, visit :
http://thekiransblog.blogspot.com/2010/02/multithreading.html
(http://thekiransblog.blogspot.com/2010/02/multithreading.html)

Thanks, very useful guide. (/article/1363#comment-332354)
Submitted by Jyaan (not verified) on Sun, 02/01/2009 - 05:27.

Thanks, very useful guide.

Re: What Is Multi-Threading? (/article/1363#comment-273)
Submitted by Anonymous on Fri, 01/23/2004 - 03:00.

I teach English HighSchool Computer students, the exam board sneakily put multi-threading on the exam paper this January. This article clarified the whole thing. THANKS. regards: Mel Walker, Durham Sixth Form Centre, England.