Pre-emptible Linux
A REALITY CHECK

Can Linux be a "real-time operating system"? Kevin Morgan investigates

Real-time is a term that characterises a particular application. Hard real-time means an application fails catastrophically if deadline requirements are not met. Soft real-time means an application suffers degradation in quality, but not catastrophic failure, if deadline requirements are not met. Both hard and soft real-time are clock time independent.

Linux is capable of meeting a wide variety of real-time requirements, in terms of specific timing needs, addressed by specific levels of software (interrupt service routine versus user application level). Interrupt service routine software is delayed by interrupt off periods in the kernel, and the Linux 2.4 kernel has very short interrupt off timings, with none greater than 60 microseconds on an 800 MHz Pentium III class system. This level of performance meets the vast majority of real-time requirements for interrupt level software. This is particularly true given modern system designs, where extremely fast I/O response requirements tend to be serviced by dedicated hardware in the form of intelligent I/O controllers, dedicated micro-controllers or custom dedicated hardware.

In the rare remaining cases where Linux interrupt off periods cannot be tolerated, RTLinux and RTAI are available. These are sub-kernel technologies that provide simple multi-threaded interrupt handling environments for driver level software. These environments emulate (virtualise) interrupt management requests from Linux, and thereby reduce the worst case interrupt off timings for the driver software written for these environments from the 60 microsecond level down to approximately 10 microseconds.

Modern real-time environments typically involve substantial control and monitoring software in the real-time control path. Such software resides at the user application level. For example, consider real-time control software written in Java, running on a JVM, an increasingly common design choice. Such a system would never be structured as driver level software. Response requirements for applications are directly tied to the kernel's ability to pre-empt a running process and switch to a higher priority process (newly awoken) very quickly. The lack of kernel pre-emption in Linux means that long system calls can delay high priority user process execution for relatively long periods, running into the tens of milliseconds in a 2.4 kernel. There is now a kernel pre-emption patch that today reduces this time down to one to two milliseconds, with further improvements planned for the future.

Whether an operating system capable of these levels of responsiveness guarantees is considered real-time or not is a positioning rather than a technical issue. This level of improvement in Linux moves it from "problematic" to "very acceptable" for the vast majority of applications that have real-time requirements (soft or hard).

Maintenance cost and longevity

All of the changes for the pre-emptible kernel patch directly leverage the SMP spinlocks, which are themselves fundamental in Linux for symmetric multiprocessing. The code modifications in the pre-emptible kernel patch are thereby limited to the four areas (see The Patch Specifics). New kernel code that functions correctly in an SMP kernel requires absolutely no additional changes in the pre-emptible

The patch specifics

The pre-emptible kernel patch modifies the definition (implementation) of a spinlock, changing it from its symmetric multiprocessing (SMP) specific implementation to a pre-emption lock. In both cases, the locking function acts as a control on re-entrancy to a critical section of kernel software. Additionally, the pre-emptible kernel patch modifies the interrupt handling software to allow rescheduling on return from interrupt if a higher priority process has become executable, even if the interrupted process was running in kernel mode (provided the process is not in a critical pre-emption locked region). Spin unlocks are redefined to return the system to a pre-emptible state, and check if an immediate context switch is needed. Lastly, the kernel build definition for a uniprocessor target system is modified to include the spinlocks (implemented as pre-emption locks). Through these four basic changes, the Linux kernel becomes generally pre-emptible (with short non-pre-emptible regions corresponding to the spinlocked regions in an SMP kernel). Process level responsiveness is dramatically improved, both on average and in the worst cases.

kernel patch. Thus, maintenance of the patch against the evolving Linux base is low cost.

Improvement in Linux process level responsiveness is a must requirement for many embedded system designers considering the use of Linux as an OS platform. Embedded developers have a simple choice: enable kernel pre-emption if needed by the demands of their responsiveness requirement, or continue to use non-pre-emptible Linux if sufficient as is.

Audio processing under load

In order to achieve over 20x improvements in process level responsiveness, what level of throughput loss is acceptable? If throughput loss is less than two to three per cent, the cost is outweighed by the improvement in system responsiveness. This trade off does not have to be made if every ounce of throughput is critical, and process level responsiveness is not. Users can select pre-emption or not, as they see fit.

Official Linux kernel source

Pre-emptible kernel technology (as a build option, similar to SMP) should be included in the Linux source code, as provided at kernel.org, starting with the 2.5 kernel base. It is a fundamental improvement in Linux, which has value to all Linux user communities (desktop, server and embedded), and should be provided with this central distribution.

However, continued development and deployment of pre-emption technology in Linux will not slow down if the technology is not integrated into the official source tree. Many Linux technologies are available and in widespread use today that are not part of the source code, and may never be included. This is one of the key benefits of open source; new and innovative technologies can be developed and provided when necessary, with the best getting extensive usage and support. Independent of Linux 2.5 and beyond, Linux kernel pre-emption technology is available today as an open source patch and there will continue to be enhancements to this technology.

In any case, certain embedded Linux companies are committed to the long-term availability and support of this capability and committed to providing this leading edge advancement across all major target platforms. Providing an alternative semaphore implementation that utilises priority inheritance is an improvement under design. Continuing to refine long spinlock held regions is an ongoing effort. Characterising throughput impacts (positively and negatively) on a number of workloads is under progress and will be shortly available. A number of application success stories will become public over the course of the next year as this technology is widely designed in and deployed by embedded system product organizations.

The impact of throughput

At a simplistic level, changing a uniprocessor kernel to add internal re-entrancy management means "more code" and hence "more time." Superficially, a pre-emptible kernel will have reduced throughput.

At the heart of the throughput issue is the question of a balanced system design, and the overall design objectives. How important is a responsive Linux? In a world of streaming media, responsiveness is quite important. A demonstration of the pre-emptible kernel doing simple audio processing shows that even a trivial load on non-pre-emptible Linux causes user process delays that exceed the threshold of the human ear, and audio glitches are heard. With pre-emption enabled, these delays are vastly reduced, and no audible glitches are heard.

Throughput concerns

Some oppose a pre-emptible kernel because of throughput concerns. Others oppose pre-emptibility because of concerns about growing complexity in the kernel. This argument is specious, because the pre-emption approach takes advantage of already required and in-place SMP locking. No additional complexity is created. All Linux kernel engineering must already take into account SMP requirements. Some oppose continued refinement of SMP locking to achieve better SMP scaling (on higher way SMP systems); such refinement has the beneficial side effect of also reducing pre-emption off periods in a pre-emptible kernel.

Pre-emptibility on 2.4 already provides dramatic improvements in user process responsiveness, and while further improvement would be beneficial, the current level of improvement is already of tremendous value. Hence, the pros and cons of improving SMP scaling in Linux can be debated relatively independently of pre-emptibility improvement opportunities.

Responsibility to the community

Embedded Linux companies have responsibilities to the open source and Linux communities, as well as to the embedded system product development communities. They have a responsibility to innovate and release innovations early and often, for public comment and contribution. They have a corporate responsibility to do their best to enable Linux to be a viable operating system platform for embedded system design and implementation. Their customers will also find significant value in the exercise of that responsibility, through the delivery of such product technologies as a pre-emptible Linux kernel.

The Linux kernel community is large and diverse. In every technical area, there is lively discussion and debate. Pre-emptible kernel technology is no different. The embedded systems marketplace, and the Linux community itself, will eventually decide the relative merits of pre-emptible kernel technology.

The author

Kevin Morgan is Vice President of Engineering at MontaVista Software. He has 20 years of experience developing embedded and real-time computer systems for Hewlett-Packard Co. Experienced in operating systems and development. Kevin was a member of the HP 1,000 computer software design team. While at Hewlett-Packard, he worked as an engineer, project manager and section manager spanning the development of five operating systems. Most recently serving as HP-UX Operating System Laboratory Manager, Kevin was responsible for overall HP-UX release planning, execution and delivery for Hewlett-Packard server computers.