3/07 June 2007

boards & solutions

the european embedded computing magazine

REPRINT FOR OSADL

Native mainline Linux: fit for embedded and real-time systems

Native mainline Linux: fit for embedded and real-time systems

By Carsten Emde, Open Source Automation Development Lab,
and Thomas Gleixner, Linutronix

When Linux was born on August 1, 1991, nobody expected that 15 years later the scale of supported hardware would range from tiny microcontrollers up to large multiprocessor mainframes, or that Linux would become a real-time operating system.

Figure 1. Linux Fedora 7 system running a total of seven virtual systems simultaneously

What is real-time and how is it measured? As generally agreed, it is not an average fast response that makes a system real-time compliant but the reliability that a defined response interval is not exceeded under whatever circumstances. A system, for example, that reliably responds to external signals within 30 minutes is a real-time system whereas another system, that relies on a timely response within 30 microseconds but fails once in a year, is not.

In order to determine the real-time capabilities of a system, an external signal is applied repeatedly to a controller input such as a digital input line or a handshake line of a serial or parallel communication controller. The analysis requires that a few lines of code are inserted into the beginning of the respective interrupt service routine and into the initial part of a user space program waiting for this interrupt. These lines of codes are written in such a way that a pulse is generated at an output device. This pulse can then be measured along with the input trigger using an oscilloscope. In addition, the respective interrupt line is also monitored.

The interval between the input trigger and the corresponding level change of the interrupt line is called "gate latency" or "controller latency". The interval between the input trigger and the start of the interrupt service routine is called "interrupt latency", and the interval between the input trigger and the execution of the user space program is called "overall latency", "preemption latency", or simply "latency" (figure 2). Normally, a large number of single measurements is taken over several hours while generating a realistic system load, and the longest measured overall latency is taken as the final result. This is then called "worst-case latency". To gain a better overview, the data points are best displayed graphically in a latency histogram (figure 3).

Initially, the Linux operating system was adopted primarily as a server platform - probably due to its support of generally available and inexpensive hardware in combination with its superior network capabilities and run-time stability. At a later stage, support for architectures other than x86 was added so the use of Linux could have been extended immediately to embedded systems. However, embedded systems often require real-time capabilities which Linux as a server operating system was never intended to provide. In fact, in a 2001 survey published in April 2002 as a Venture Development Corporation Whitepaper (http://linuxdevices.com/articles/AT6328992055.html), the most important factor inhibiting Linux adoption by respondents planning to use Linux was the item "real-time limitations". At this time, it was a common understanding that a real-time operating system must be conceived from the very beginning as such and that every single component must be designed with all aspects of real-time execution in mind. Thus, retrofitting real-time capabilities into the native mainline Linux was deemed impossible. As a consequence, several Linux real-time projects were started that used the dual-kernel approach. This approach replaces the Linux kernel by a small real-time kernel to which all interrupts are sent and which controls the entire system. One of its tasks is a modified Linux kernel; this enables execution of Linux user space programs and thus provides binary Linux compatibility of the entire system (figure 4).

Examples of solutions using this approach are RTLinux, RTAI, and Xenomai; they have been used repeatedly and successfully in recent years in a large number of industrial applications. RTLinux, RTAI, and Xenomai offer, similarly to the RT-Preempt patches, a POSIX compatible API to access real-time related functionality. Xenomai, in addition, comes with a collection of other APIs (called "skins"). They greatly facilitate the migration from proprietary real-time operating systems. In the foreseeable future, it is

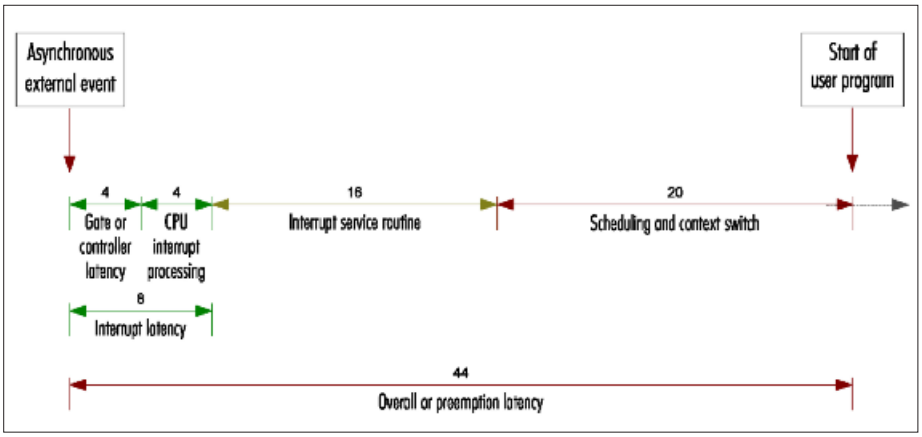

Figure 2. The various elements of the overall or pre-emption latency

well conceivable that Xenomai will adopt the RT-Preempt approach explained below and provide additional functionality on top of the real-time capable mainline Linux kernel. In parallel, the mainline Linux evolved as well. A first important step was to replace the O(n) scheduler by Ingo Molnar's O(1) scheduler. This new scheduler was already merged into the development Linux 2.5 kernel tree in 2002 and became the standard scheduler in Linux 2.6 introduced in December 2003. It ensures that the time needed to schedule is fixed and deterministic irrespective of the number of active tasks - an important prerequisite towards real-time performance. Another important step which was also added to the development kernel 2.5 and became readily available in 2.6 was to enable scheduling during execution of certain parts of the kernel. This feature is called kernel pre-emption; it prevents a low-priority process from blocking a high-priority process while the former is executing a kernel call. Initially, only small portions of the kernel code were made interruptible but they have been extended continuously.

Due to the very nature of the kernel tasks, there will always be a certain part of the kernel code that must be executed exclusively, and thus requires locking against other processes. Such *mut*ual *ex*clusion is done using so-named mutexes; but a general problem is, again, that a low-priority process executing a kernel function under locking conditions may block a high-priority process that is about to execute the same kernel function. This phenomenon is called priority inversion, and one of the solutions is to bequeath the priority level to the mutex and let it execute under the same priority as the calling process. This special mutex variant that is called Priority Inheritance Mutex (abbreviated PI Mutex) was implemented for use in the Linux kernel.

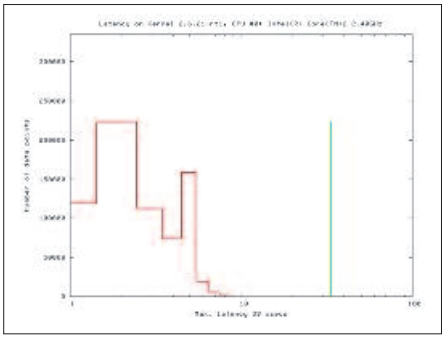

Figure 3. Example of latency data presentation in a histogram. The vertical green bar marks the worst-case latency.

In addition to kernel code, drivers and specifically their interrupt service routine may also cause latencies. In consequence, to further reduce the overall latency, a mechanism must be found to enable scheduling during execution of the interrupt service routines. This may not be meaningful during its initial part when hardware is accessed more frequently, e.g. to acknowledge the interrupt and to read and write the various data and status registers of a particular controller. But it may be very effective during the remaining stages of the interrupt service routine where hardware accesses occur less frequently. This feature is called interrupt threading.

Precise timing in real-time systems requires timers with a resolution in the range of microseconds. The Linux system clock, however, could only be set to an interval of 1, 4 or 10 milliseconds corresponding to 1000, 250 and 100 Hz, respectively. Several coding attempts to increase the system clock failed, since clock frequencies above 1000 Hz did not scale. It was, therefore, decided, to separate the timers needed for high-resolution timing purposes from the system clock timers and to introduce a new feature called high-resolution timers ("hrtimer" subsystem). This subsystem now also contains the feature "dynamic tick"; it allows to completely disable the periodic tick when the system is idle and is used to safe battery power in mobile systems. Until kernel release 2.6.18, only the O(1) scheduler and a certain level of kernel pre-emption were part of the mainline kernel.

The other features, such as PI mutexes, high-resolution timers and interrupt threading, were only available when the kernel was patched with the "RT-Preempt" patch downloaded from Ingo Molnar's repository at RedHat http://people.redhat.com/mingo/realtime-preempt/. The various elements of this patch described above were developed over the last couple of years by Ingo Molnar and Thomas Gleixner with support from many kernel developers worldwide. There was always hope that the RT-Preempt patch would be integrated into mainline Linux one day. But convincing Linus Torvalds that making Linux a real-time operating system would do no harm, but greatly enhance its usability, was not an easy task. Fortunately, there were an increasing number of arguments to make Linux real-time compliant - not only for industrial automation projects but also for standard server and desktop Linux systems:

- ✓ Audio and video mixing and recording
- ✓ Reliable timestamps in financial and commercial transactions
- ✓ Voice over IP and streaming video services.

At the occasion of the Ottawa kernel summit in July 2006, Linus Torvalds finally accepted merging the first part of the RT-Preempt patches and agreed to gradually enter the other components into mainline. As of kernel version 2.6.21, approximately 60% of the RT-Preempt patches (complete kernel preemption, PI mutexes and the high-resolution timers) are already available in mainline Linux. It is expected that most of the rest will have followed until the end of 2007.

Linux goes Virtualisation

Virtualisation of an entire computer requires an additional level of privilege (the "hypervisor" privilege) to ensure that the virtual machine monitor can control both the system and user level of a virtual machine and also that the virtual machine cannot interfere with the activities of the host system or another virtual machine. Up to about one year ago, this additional privilege level was realised mainly in software such as products provided by VMware Inc. (Palo Alto, USA). However, most of the newer x86 processors from Intel and AMD provide the required hypervisor privilege level in hardware which greatly simplifies the design of the virtual machine monitor. The related CPU flag is called vmx (Virtual Machine eXtension) in Intel and svm (Secure Virtual Machine) in AMD processors, respectively.

In October 2006, Avi Kivity posted a Linux character device driver that allows to manage a virtual system by means of I/O control commands. In addition, he provided a modified version of the qemu emulator where the emulator was replaced by the appropriate kvm commands (http://sourceforge.net/projects/kvm/). The orig-

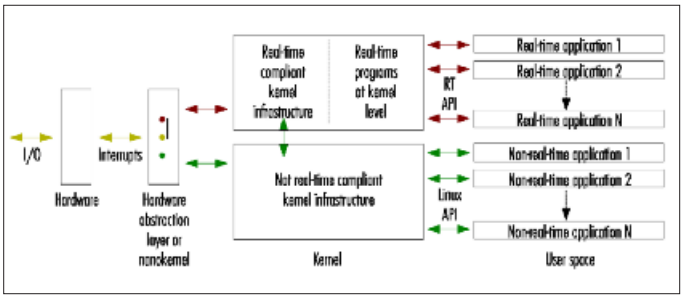

Figure 4. The "dual-kernel approach" to realize a real-time Linux system

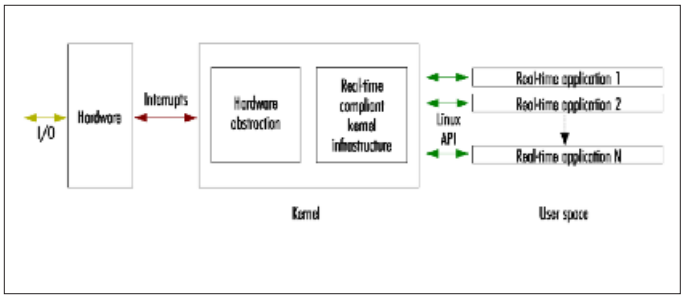

Figure 5. The "single-kernel approach" to realize a real-time Linux system

inal qemu emulator was developed by Fabrice Bellard (http://fabrice.bellard.free.fr/qemu/). Probably because of its elegant design, excellent code quality and minimum interference with the kernel, Linus Torvalds decided only two months after its initial submission to merge kvm into mainline Linux, and it was first released in kernel version 2.6.20 on February 4, 2007.

An example of a Linux system that uses the kvm driver and the qemu emulator to simultaneously run a total of seven virtual systems is given in Figure 1. None of the virtual systems needed any modification prior to be installed under kvm. The performance of the virtual systems is not different from that of the host system as long as straight CPU code is executed, e.g. during execution of Dhrystone or Whetstone benchmarks. I/O operations, however, have a reduced performance, since the data must be transfered twice - first from the virtual system via the virtual PCI bus to the host system and then from the host system through the physical PCI bus to the peripheral device. The hard disk transfer rate of the system shown in figure 1, for example, amounts to about 68 MByte/s in the physical Fedora system and to about 36 MByte/s in the virtual Fedora systems.

In comparison to other virtualisation solutions, kvm has the advantage that it is part of mainline Linux, and will therefore be continuously developed and adapted to other kernel features. Of special interest is a real-time system running a kvm virtual system with preserved real-time capabilities of the host system. This would allow to use a single computer hardware for machine control and graphical user interface - even if the two functions require two different operating systems. As a consequence, system reliability would increase while costs would decrease.

The advent of open source software makes it possible for the first time for different and even competitive companies to share their efforts and develop together base components of the operating system. This common effort, however, could be more effective if an organisation were available to synchronise these activities. The Open Source Automation Development Lab (OSADL) was founded for this purpose (http://www.osadl.org). OSADL member companies are active in the field of industrial automation such as machine, machine tool and equipment manufacturers, computer hardware and software manufacturers and software service providers. The membership fee is primarily used to delegate the development of machine and automation software components requested by a majority of the OSADL members. Current projects, among others, are: Linux RT-Preempt patches, industrial I/O framework, migration tools, upstream submission of Linux kernel components, making kvm real-time compliant, real-time Ethernet drivers for Linux. ■