Resource control in RT-Linux
(A short position paper)

Victor Yodaiken
Department of Computer Science
New Mexico Institute of Technology
Socorro, NM
`yodaiken@cs.nmt.edu`
`http://luz.nmt.edu/~rtlinux`

RT-Linux is an operating system in which a small real-time kernel coexists with the Posix-like Linux kernel. It is enormously convenient to have a single machine that supports both hard-real-time tasks and all the services of a standard operating system. In fact, over the last decade, several of the most widely used real-time operating systems such as QNX and Vxworks, have added servers or modules to provide everything from TCP/IP to sophisticated GUIs. An alternative approach leads one to add real-time schedulers, page-locking, and other real-time facilities to an existing operating system. RT-Linux uses a third method. RT-Linux runs Linux as a task under a small real-time executive. Linux still provides operating system services to ordinary, non-real-time processes and still directly manages most system devices — such as the disk drives and video hardware. But Linux is prevented from disabling interrupts. All interrupts are initially handled by the Real-Time kernel and are passed to Linux only when there are no real-time tasks to run. To minimize changes in the Linux kernel, it is provided with an emulation of the interrupt control hardware. Thus, when Linux has "disabled" interrupts, the emulation software will queue interrupts that have been passed on by the Real-Time kernel. In the current version of the system, real-time tasks communicate with Linux processes via a lock-free queues.

In practice, the RT-Linux approach has proven to be very successful. One user reports that hisRT-Linux application was able to poll an A/D board reliably at a rate 30% faster than a commercial program was able to do on the same hardware under DOS — an operating system that imposes no overhead at all. In both experiments, the polled data was logged to disk and displayed. In the RT-Linux case the display was through Motif.

This difference in speeds is consistent with other reports: at the worst, we see no performance penalty. While some of the difference may come from more efficient operation of the low-level code, we believe that much of the difference can be attributed to performance on the data logging and display due to the average-case efficiencies of the Linux OS with its quite sophisticated file buffering, memory management, and I/O handling.

Real-time tasks under RT-Linux are relatively primitive. These tasks have statically allocated memory, rely on a simple pre-emptive scheduler with fixed priorities, and have no connection to more sophisticated services except via the statically allocated lock-free fifo channels. The Linux OS takes care of all but the time-critical OS functions. The real-time tasks and the real-time scheduler are even loaded into memory by Linux as loadable kernel modules. Indeed, our reliance on loadable kernel modules allows us to experiment with different task sets and schedulers without interrupting non-real-time operations.

Although we plan to develop alternative schedulers and replace the fifo-channels with a more flexible interface, our intention is to keep the real-time components simple, fast, and potentially verifiable, by pushing all complex resource allocation issues into the non-real-time system. In particular, we hope to be able to factor QOS problems into hard real-time components that collect or distribute time sensitive data, and Linux processes or threads that monitor data rates, negotiate for process time, and adjust algorithms.