



Real-Time Linux®: The RedHawk™ Approach

Jason Baietto

Chief Systems Architect

Joe Korty

Sr. Consulting Engineer

John Blackwood

Sr. Consulting Engineer

Jim Houston

Consulting Engineer

Concurrent Computer Corporation

ABSTRACT

Read this white paper to learn about Concurrent's evolution to a commercially available Linux RTOS that safeguards deterministic behavior while minimizing any impact to system stability and complexity. At the heart of Concurrent's approach to real-time Linux is the concept of processor shielding, a means to reserve CPUs for real-time processes. Contrasts to standard Linux and alternate industry proposed processor shielding strategies (PREEMPT_RT) are discussed.

INTRODUCTION

Concurrent Computer Corporation has been selling real-time solutions in various forms for over 40 years. While the majority of Concurrent's past real-time solutions were proprietary, Concurrent fully embraced free and open source software in 2000 and shifted focus to a new real-time product based on GNU/Linux called RedHawk Linux. A key goal of RedHawk Linux is to provide the same level of real-time support that Concurrent's existing customer base depends upon, while leveraging open source software with minimal modifications in order to stay as close as possible to the kernel.org source code. The RedHawk Linux solution meets Concurrent customer requirements and allows for continuous Linux updates by providing features and software layers that isolate applications from the dramatic level of change occurring in the Linux community.

OVERVIEW

RedHawk is a high-performance Linux distribution. At the core of the product is a real-time enhanced Linux kernel that can in theory be integrated with any Linux distribution. The RedHawk kernel is based on pure stable kernel releases from *kernel.org* and is then enhanced by including various open source patches as well as locally developed features. Concurrent frequently re-bases RedHawk to the latest Linux kernel releases in order to provide community fixes and enhancements to customers. Concurrent closely follows the versions of the various Linux distributions they release, ensuring that support for both the kernel and Linux distributions is as up-to-date as possible.

RedHawk Linux contains a set of user-level packages including libraries, headers, documentation and commands that provide access to the kernel's real-time features. The APIs provided are an evolution of the APIs that have been developed during Concurrent's long histo-

ry of providing real-time solutions. These APIs provide real-time application developers with a feature rich, stable and time proven interface. The use of this interface has allowed many of Concurrent's customers to move their applications from Concurrent's older proprietary solutions to RedHawk, and then move again from a RedHawk based on the 2.4 Linux kernel series to a RedHawk based on the 2.6 Linux kernel series, with very minimal code changes. In addition, Concurrent has many tools to help easily migrate users to RedHawk from other platforms.

Many of Concurrent's customers wish to develop and deploy real-time applications written in the Ada language, still widely used by the United States military. Concurrent's *MAXAda™* environment provides a feature rich Ada development environment that takes full advantage of the real-time enhanced kernel included with RedHawk Linux. Concurrent provides a high-performance FORTRAN development environment, along with the standard GNU C and GNU C++ development environments.

An optional tool suite called the *NightStar™Tools* is available and is a big part of the value-add of RedHawk. The NightStar Tools provide debugging and analysis tools that are specifically targeted to real-time application development and system tuning, again taking full advantage of the real-time enhanced kernel included with RedHawk Linux. Note that the NightStar Tools can debug, analyze and tune applications written in C, C++, Ada and FORTRAN.

Concurrent enhances RedHawk's value via hardware integration, custom engineering services, maintenance contracts and extensive end-user documentation. Training at all levels, from beginning Linux programming to advanced real-time programming techniques on RedHawk, is available at the home office or at the customer's site.

PROCESSOR SHIELDING

At the heart of Concurrent's approach to real-time is the concept of processor shielding, which is the reservation of specific CPUs in the system for real-time processes. For processor shielding to work there obviously must be more than one CPU in the system, and for nearly three decades Concurrent's hardware solutions have consisted of sophisticated SMP architectures often with *non-uniform memory architecture* (NUMA) support. Now that multi-core processors are widely available, the number of platforms that can exploit shielding has exploded, and this has enabled inexpensive single processor

socket computers to become highly effective real-time solutions.

The RedHawk kernel combines existing shielding support in the Linux kernel with locally developed enhancements and provides convenient user-level tools to configure and control processor shielding. For example, the supplied *shield* command can be used to dynamically block processes, generic interrupts and even the local timer interrupt from occurring on a specified set of CPUs. Real-time processes running on a dedicated CPU can be ensured of the full bandwidth of the processor without interruption, and importantly, with virtually no cache interference, dramatically improving the process' determinism. No other real-time mechanism can offer this level of determinism.

It is important to realize that Concurrent's processor shielding implementation consists of a relatively small set of kernel extensions and does not involve a dramatic redesign of the entire kernel spin-lock and interrupt model (as is the case with the proposed PREEMPT_RT patch being developed by Ingo Molnar). This low-overhead approach dramatically enhances the viability of the solution and minimizes any impact to system stability and complexity, something that has not been the case up to this point with the PREEMPT_RT approach.

PREEMPTION REDUCTION

RedHawk provides a user-space mechanism that allows a running process to tell the kernel that it cannot be preempted for a brief period of time. This feature is implemented by having the process register a special memory location with the kernel called a *rescheduling variable*. The process can then control preemption simply by writing to or clearing the rescheduling variable, without requiring any system calls to be performed.

This feature is used extensively throughout the user-level libraries provided with RedHawk, and in particular it is used to provide a reliable and fast implementation of userlevel spin locks.

INTERRUPT REDUCTION

RedHawk Linux has been extensively tuned to minimize the number of interrupts that occur on a CPU during the execution of a real-time process. This tuning has been done in various ways which are discussed in the following subsections.

Interrupt Blocking

Real-time applications under RedHawk have the ability to block delivery of all interrupts, for short periods of time, to the CPU that the application/thread is running on. In standard Linux, this ability to block interrupts is available only to kernel code, such as device drivers.

This ability is useful for those applications that have short code sequences with run times no more than a few hundred microseconds, and also have very tight timing requirements.

TLB Flush Reduction

The *vmalloc* kernel virtual address space is used for a variety of purposes, such as temporary buffer space, kernel data structures, and *ioremap* operations. When any previously allocated *vmalloc* virtual space area is freed, a *cross processor interrupt* (CPI) is issued to all the other CPUs in the system so that each CPU will issue a *translation look-aside buffer* (TLB) flush instruction to remove any cached translations to the previously used *vmalloc* virtual area.

While these types of TLB flush CPIs typically take only a short peri-

od of time to execute (3-5 microsecond range), a series of multiple back-to-back TLB flush CPIs can occur and thus greatly lengthen the total amount of time that a CPU is processing TLB flush activity. This flurry of TLB flush CPIs has been observed to produce a noticeable lack of determinism on a shielded CPU.

The RedHawk *vmalloc* support code has been rewritten to provide a configurable, alternative *vmalloc* algorithm that greatly reduces the number of *vmalloc*-related TLB flush CPIs, thus eliminating this source of potential jitter on shielded CPUs.

The *vmalloc* support has been modified so that when new *vmalloc* space is allocated, a kernel virtual address that is higher than the last *vmalloc* allocation is selected, whenever possible. Only when the search for free space wraps around to the beginning of the *vmalloc* virtual space area are TLB flush CPIs issued to remove any previously used translations, thus ensuring the impact to determinism is small and controlled.

Memory Preallocation

RedHawk includes graphics drivers that have been enhanced to minimize the need to cause CPIs. Proprietary drivers such as the popular drivers from NVIDIA® have been enhanced and are included.

The interrupt reduction has largely been achieved by providing a mechanism to preallocate graphics pages at boot time, so that floods of TLB flushes will not need to be performed as new pages are mapped by the graphics driver.

LATENCY MINIMIZATION

A fast, timely reaction to external events is generally what defines a real-time system. A system which exhibits this property is said to be a low-latency system. Areas that can cause poor latency are:

- Frequency scaling damages latency because it takes time for the OS to detect the need to increase the frequency of some slowed-down CPU. It then must *step up* the frequency in the proper pattern required, up to the final, normal high frequency of operation. During this step-up time, the CPU is not running at full capacity and hence is responding slower to the external event than it would have if it had already been running at full speed.
- Some multi-core chips have an additional power saving state called the C1 *extended* (C1E) state. This state is entered when all the CPUs on some core are in the *halt* instruction. In this state everything on the chip turns off, even the ability of the CPUs on that chip to generate a local timer interrupt. Thus, if the C1E state is allowed to happen, the OS must turn off high precision POSIX™ timers in favor of the HZ-based low precision POSIX timer system, as high-precision timers require the use of the (now disabled) local timer interrupt for their proper operation.

In order to preserve the desired low-latency characteristics of the system, RedHawk has code that prevents it from changing clock frequency and from entering sleep states. In order to ensure that the local timer interrupt is reliable, RedHawk ensures it never allows the C1E state to be entered.

Eliminating the effects of power management has the additional benefit of ensuring a highly reliable *time stamp counter* (TSC) CPU register which is essential for real-time systems, yet which is something that standard Linux does not even attempt to provide.

TIME-SOURCE IMPROVEMENTS

Real-time applications generally need a time-source that is:

1. Extremely fast to access
2. Extremely high-resolution with at least nanosecond resolution
3. Extremely accurate to within a few hundred nanoseconds of the correct time

Standard Linux does not provide a time-source which meets all three of the above criteria. The closest provided is the standard clock-source based on the per-CPU TSC register. The TSC is fast to access and also returns a precise number, but too easily drifts away from the correct time, thus losing accuracy.

Alternatively, a high-quality externally supplied time-source, for example a GPS module attached to the PCI I/O bus, may be extremely accurate internally but suffers from slow and varying access times due to the relatively long distance that devices on the PCI bus are away from the CPU. This slow and varying access is a function of the varying traffic load on the PCI I/O bus, and on the memory bus to which the PCI I/O bus is attached.

RedHawk Linux solves these problems by implementing the concept of *stacked clock-sources*. Applications still use the `clock_gettime(2)` system service to fetch timestamps. However, instead of internally having only a single clocksource driving the service, there is a pair of attached clocksources. The primary clock-source of the pair has to have the attribute of being fast to access and of returning a high precision value. The TSC has these attributes so it is universally selected as the primary clock-source. The secondary clock-source has to have the attribute of being extremely precise and extremely accurate, but it need not have the attribute of fast and unvarying access times.

The primary clock-source is always used by `clock_gettime(2)` to return the system time. RedHawk uses the secondary clock-source to periodically recalibrate the primary. This recalibration keeps the primary from drifting too far away from the correct time. The recalibration algorithm takes into account the length of time it takes to read the secondary clock-source and has a compensation mechanism that minimizes the effects of the variation in read time.

For RedHawk, the ideal secondary clock-source is Concurrent's locally developed *Real-Time Clock & Interrupt Module (RCIM)*. More information about this card is discussed in the "Hardware Certification" section below.

PROCESS SYNCHRONIZATION

RedHawk provides a fast and efficient sleep/wake mechanism that can be used between a group of cooperating threads or processes. This service is based upon the open source Post-Wait patch developed by SGI®, however it has been enhanced to take advantage of rescheduling variables to ensure atomic wake-ups and improve determinism.

ADDITIONAL SCHEDULERS

RedHawk provides the standard Linux process scheduler as well as a Concurrent developed *frequency based scheduler (FBS)*. The FBS is a high-resolution task scheduler that enables the user to run processes in cyclical execution patterns. The FBS controls the periodic execution of multiple, coordinated processes utilizing major and minor cycles with overrun detection. A *performance monitor (PM)* is provided to view CPU utilization during each scheduled execution frame.

The FBS interface has been developed and enhanced over decades and now represents an ideal scheduler optimized for classic hard real-time applications such as data acquisition, simulation and process control. Applications written for the FBS have required little or no change over the last several years, whereas in that same time the Linux scheduler has seen many re-writes with subtle changes to both load balancing heuristics and scheduler fairness.

NUMA OPTIMIZATION

RedHawk systems that use AMD Opteron™ processors or Intel® Nehalem™ or Intel Tukwila™ processors can take full advantage of the architecture's NUMA facilities. Concurrent has enhanced the `run` command to allow the specification of memory policies along with other process environment settings, creating a single tool which can conveniently be used to bind a process to both a specific processor (or core) and a specific NUMA node.

Concurrent has enhanced the `shmconfig` command to allow the specification of memory policies that control the NUMA node location of shared memory areas. The entire shared memory area may be assigned the same memory policy, or differing memory policies may be assigned to various address ranges, thus placing specific pages of the shared memory area into the NUMA node memory where it is most frequently accessed.

Concurrent provides the ability to memory shield one or more NUMA nodes with the `shield` command. When a NUMA node becomes memory shielded, the kernel will automatically migrate user pages between nodes so that only user pages belonging to processes that are biased to execute on CPUs in the memory shielded NUMA node will be contained in that node's memory. Thus, process determinism is improved by eliminating remote memory accesses from processes executing on the memory shielded NUMA node, and by eliminating remote memory accesses into the memory shielded NUMA node by processes executing on non-shielded NUMA nodes.

In addition to automatic page migrations that occur when a node becomes memory shielded, a kernel feature called *page replication* (based on a patch developed by Nick Piggins) is used to create multiple copies of the same text and read-only data pages in multiple NUMA nodes, thus keeping memory accesses local for memory objects such as shared library pages, which are typically referenced by many processes executing across multiple nodes in the system.

When a non-memory shielded NUMA node's memory becomes full, page allocation requests may overflow into another NUMA node's memory. In this case, a memory shielded NUMA node's memory could end up containing pages belonging to non-memory shielded tasks. To prevent this occurrence, when a NUMA node becomes memory shielded, the kernel automatically reorders the `zone lists` so that page overflow allocations will be first satisfied using pages from non-memory shielded nodes, with memory-shielded node memory being used only as a last resort when all non-memory shielded nodes are low on memory.

Note that memory shielding may be combined with process, interrupt and local timer shielding to create a *totally* isolated NUMA node dedicated to real-time processing.

As an aid for application development, the Concurrent `run` command may be used to view the number of currently mapped user pages in each NUMA node on a per-process basis for all user-space processes in the system, or for a specific set of processes. Additionally, the `numapgs` command provides the NUMA node location of each page

currently mapped into the user address space of a single process, as well as flags indicating whether each page is replicated or user-locked (*mlocked*).

HARDWARE CERTIFICATION

Concurrent is committed to using *commercial off-the-shelf (COTS)* hardware in real-time solutions whenever possible. A large percentage of COTS hardware is not suitable for real-time use for several reasons, including:

- The system BIOS does not allow fine-grained control over PCI slot IRQ assignment
- The system has devices which cannot be disabled yet are hard-wired to share IRQs with PCI slots
- The firmware generates periodic SMI interrupts which cannot be disabled
- The hardware lacks a mechanism to generate an NMI for debugging a hung system
- Some chip-sets have inherent issues with real-time (e.g. unfair memory access across CPUs)

Systems qualified by Concurrent to be real-time compliant are iHawk™ real-time systems and ImaGen™ real-time visual systems—both included in the list of systems approved for real-time solutions. Qualified systems also become integrated into Concurrent's *automated nightly test system (ANTS)*. New versions of existing iHawk systems are periodically re-qualified to ensure that subsequent board-level changes and new chip-set revisions do not impact real-time performance.

The list of qualified iHawk/ImaGen COTS systems includes offerings from many vendors including Dell™, Supermicro™ and Tyan™. RedHawk is also available on various IBM BladeCenter™ blade configurations.

Many of Concurrent's customers have a long history of using VME-based technologies, and thus RedHawk fully embraces the VME standard. Concurrent offers a PCI-to-VME chassis which can be used to host many different standard VME I/O boards, connected by fiber optic cables resulting in exceptional run-time performance. In addition, RedHawk has been ported to several different VME-based boards running multi-core processors, and thus customers who wish to use VME technology can choose between two different approaches: native VME or a VME I/O bridge from an iHawk PCI-based platform. RedHawk offers many real-time optimized drivers for industry-standard VME cards, including analog-to-digital, digital-to-analog, discrete I/O, MIL-STD-1553 (avionics) and CAN bus (automotive).

Almost all iHawk systems shipped by Concurrent contain a version of Concurrent's locally developed *Real-Time Clock & Interrupt Module (RCIM)* which is available in PCI, PCI Express and PMC form factors. The RCIM provides connections for external device interrupts, programmable interrupt generators and real-time clock timers that can trigger interrupts. Multiple RCIMs can be cabled together to provide distributed interrupts allowing the RedHawk FBS to synchronize processes across multiple machines with very high accuracy. The RCIM provides a mechanism to synchronize system clocks and it can optionally be fitted with a GPS receiver and an oven-controlled crystal oscillator for extremely high precision.

DEVELOPMENT ASSISTANCE

In addition to providing unmatched real-time performance, RedHawk is specifically designed to simplify both real-time application and kernel driver development.

To facilitate real-time application development, the kernels included in RedHawk have been instrumented with a virtually lock-less very low-intrusion *kernel trace* mechanism. Using Concurrent's NightTrace™ tool, kernel traces can be combined with user-level application traces to provide a highly detailed picture of both kernel and application events displayed on the same time-line. This provides unprecedented visibility into how the real-time application and the kernel interact, and is a tremendous aid in design, tuning and debugging.

Note that Concurrent's kernel trace mechanism is implemented by inserting permanent trace point macros at specific key places inside the kernel code. These macros can be conditionally disabled, and RedHawk provides pre-built kernels with and without kernel trace. It is important to understand that the kernel trace mechanism is primarily intended to be used for the analysis of real-time application behavior; in particular, it is not designed for generic kernel debugging and is not intended to be an ad-hoc kernel debugging tool like Kprobes. Conversely, Kprobes would be a poor mechanism to use to implement kernel trace for many reasons:

- Kprobes is not designed to collect large volumes of trace data efficiently; it uses int3 traps, duplicates large portions of stack and locks a global hash table to look up unique handler functions causing excessive SMP contention.
- Kprobes is inflexible; it has limitations regarding where probes can be placed, and is limited to accessing only data that exists in registers at the time of the probe.
- Kprobes is inconvenient, especially for end users; it requires the compilation and loading of a kernel module each time a configuration change is made.
- Kprobes requires a high level of maintenance; maintaining a set of accurate trace points across evolving kernel versions would be very difficult.

Thus, while Kprobes may excel as a generic ad-hoc kernel debugging tool, it is unsuitable for the basis of a mechanism primarily intended to gather exhaustive kernel event details for the purpose of illuminating real-time application behavior.

The NightStar tools include the NightView™ real-time optimized application debugger and the NightTune™ real-time application tuning utility. The combination of NightTrace, NightView and NightTune results in an integrated suite of tools custom tailored for real-time application development and tuning.

To facilitate kernel driver development, RedHawk includes support for the KDB and KGDB kernel debuggers. In addition, emphasis is placed on tools to create and analyze crash dumps from customer sites, and RedHawk has supported LKCD and now the kexec/kdump method of producing crash files.

RedHawk makes the standard *user-level device driver (UIO)* layer available, complete with documented working example user-level driver code, to further simplify kernel driver development.

COMMUNITY INVOLVEMENT

Concurrent has been actively involved in the free software and open source communities for many years. Concurrent currently sponsors open source projects including *run*, *cpuid*, and *nuu*, and members of the Concurrent kernel development team post bug fixes and feature patches to the Linux Kernel Mailing List. Previous patch submissions have involved high-resolution timers, POSIX timers, kernel debuggers, RCU, POSIX message queues, graphics drivers, ptrace, NUMA and NFS and some of the patches have now been incorporated into the latest versions of the official Linux kernel.

FUTURE DIRECTIONS

Many more features and enhancements are planned for future versions of RedHawk, including support for using RedHawk in embedded systems, with priority always given to those features that are most requested by Concurrent's real-time customer base.

CONCLUSION

Concurrent is committed to keeping close to the changes being accepted by kernel.org, and RedHawk will make full use of the PREEMPT_RT patch if and when it has proved itself to be stable and powerful enough to be accepted into the kernel.org mainstream kernel. A RedHawk kernel that combines the best of PREEMPT_RT with Concurrent's locally developed features will provide customers with a very powerful and versatile environment for developing deterministic and low-latency real-time applications.

Concurrent RedHawk Linux is an industry-standard, real-time version of the open source Linux operating system for Intel® and AMD™-based systems. RedHawk Linux provides the guaranteed performance required in time-critical and hard real-time environments demanded by Aerospace & Defense, Automotive, Energy, Financial Services, Government, Industrial Control, Medical and Telecommunications. RedHawk Linux runs your applications when you want, where you want, and how you want, delivering the highest quality of service.

CONCURRENT A WORLDWIDE LEADER IN REAL-TIME COMPUTING INNOVATION AND SERVICE

Concurrent is a worldwide leader in real-time Linux-based computing technologies with thousands of real-time systems and software deployed globally solving the most demanding time-critical computing challenges.