# Using Preemption-Threshold Scheduling to Cut Overhead While Meeting Deadlines

By Dr. Alexander G. Dean, Center for Efficient, Secure and Reliable Computing, Dept. of Electrical and Computer Engineering, North Carolina State University

ARM®-based real-time embedded applications typically use a real-time operating system (RTOS) in order to develop a system as a collection of independent modules while maintaining responsiveness to time-critical events. A system consists of multiple tasks or threads which must run periodically or in response to some event, and which must complete their processing before a certain deadline. Real-time analysis techniques allow us to determine mathematically whether a set of threads is schedulable – whether all of a system's threads will meet their deadlines in all possible scheduling situations. This is a fundamental requirement for many embedded systems.

Most RTOSes use fully-preemptive schedulers; a thread can preempt any lower priority thread at essentially any time. Preemption allows critical processing to occur sooner than less-critical processing.

### Preemption and Processing Overhead

The responsiveness and real-time performance of a microprocessor-based system is limited by the processing overhead of context switches and interrupt service routines. The system must save the current thread's state, use the scheduler to decide which thread to run, and then restore the next thread's state. This takes a long time, even for fast ARM processors, and limits responsiveness. Over time, more demanding responsiveness requirements have forced developers to use faster, more expensive, and more power-hungry processors to accelerate context switches and interrupt response times.

Since preemptions can occur asynchronously among threads, the system needs enough RAM to satisfy all worst-case stacks simultaneously. This can be prohibitively expensive for systems with little RAM or many threads. This stack space must be large enough to accommodate worst-case function call nesting, local variable allocation, ISR activity and possibly the thread control block and context. Adding features such as a GUI, embedded web server and TCP/IP stack adds multiple threads, each of which needs its own stack. Even in systems with larger amounts of RAM, running out of memory can be a common problem.

Systems with caches suffer from cache pollution by other threads (cache-related preemption delay) unless special precautions are taken which reduce performance. As it executes, the preempting thread can evict data and instructions of the preempted thread. When the preempted thread resumes execution, it runs more slowly due to increased cache misses.

### ARM and Preemption

Over time, ARM processor architectures have evolved to reduce the processing overhead of context switches and interrupt service routines.

The processing time overhead for context switches and interrupt service routines with ARM processors has reduced the performance wall for responsive and real-time systems. The modern Cortex™-M, Cortex™-R, and Cortex™-A architectures provide improved performance, but each preemption still incurs a time overhead. Regardless of which ARM processor is used, eliminating preemptions could improve system performance, and enable the conservation of power by increasing sleep-time, or enabling the use of a lower clock rate altogether.

## Preemption-Threshold Scheduling

Preemption need not be an all-or-nothing design decision. Many systems can, in fact, meet all deadlines without full preemption among threads. In fact, there is a spectrum of possible approaches, ranging from non-preemptable to fully preemptable.

One very promising technique for limiting preemptions is Preemption-Threshold™ Scheduling (PTS)[1]. PTS was created by Bill Lamie in 1997 and is implemented in Express Logic's ThreadX® RTOS. PTS seeks to minimize the number of preemptions as much as possible, while preserving the system's schedulability. Wang and Saksena investigated the real-time characteristics of PTS, as implemented in ThreadX (Wang and Saksena 1999).

[1] Preemption-Threshold is a trademark of Express Logic, Inc.

Normal fully preemptive scheduling assigns a single priority to each thread. This priority determines two aspects of the thread's behavior: whether it can preempt another thread, and whether it can be preempted by another thread. PTS assigns a separate priority to each aspect. The nominal priority determines whether it can preempt other threads, and the preemption-threshold which is its effective priority while executing.

When the thread begins execution, its priority is automatically raised to its preemption-threshold. In this way, none of the threads with priorities less than or equal to the preemption-threshold of the executing thread can preempt it. PTS effectively creates groups of mutually non-preemptive threads that are not allowed to preempt each other.

Fully-preemptive and fully-nonpreemptive scheduling policies are special boundary cases of PTS. By assigning the preemption-threshold of each thread equal to its priority, PTS simplifies to a fully-preemptive scheduling policy. By assigning the preemption-threshold of each thread equal to the system's highest priority, PTS simplifies to a fully-nonpreemptive policy.

### Assigning Preemption-Thresholds

Preemption between threads can be limited to occur only when necessary to maintain system schedulability. Wang & Saksena developed a method to find a feasible preemption-threshold assignment if one exists. It works by starting with a default, fully-preemptive system – each thread's preemption-threshold is equal to its priority level. Starting with the lowest-priority thread, each thread's preemption-threshold is raised until any further increase would make the system unschedulable (determined using standard real-time analysis techniques).

## PTS Benefits

### PTS Reduces Number of Preemptions

Wang & Saksena have also evaluated how effectively PTS reduces the number of preemptions required in order to meet deadlines (Wang and Saksena 1999). They found that PTS reduced preemptions by 5 to 32% when compared with a fully-preemptive scheduling approach.

They evaluated randomly-generated periodic thread sets with fixed numbers of threads but uniformly distributed random periods and computation times. This approach allows us to see the behavior of different scheduling approaches across a range of workloads, and evaluate their sensitivity to different factors.

Figure 1 shows that PTS reduces the number of context switches significantly, depending on the number of threads and the MaxPeriod parameter.

Jejurikar and Gupta evaluated using PTS in a cache-based system in conjunction with dynamic voltage scaling in order to reduce system energy requirements (Jejurikar and Gupta 2004). They found that PTS significantly reduced the number of context switches required.

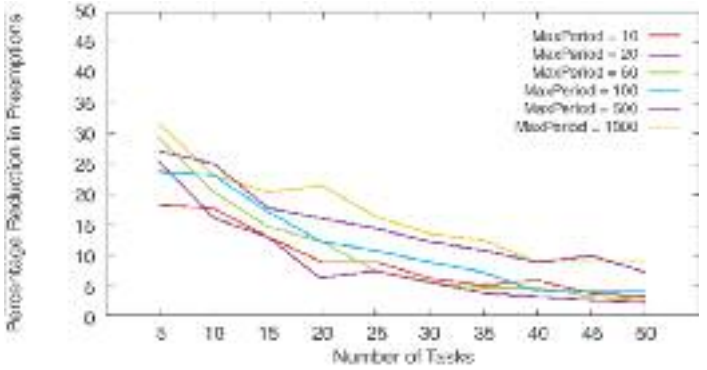


*Figure 1: Average percentage reduction in number of preemptions for PTS as compared to Pure Preemptive Scheduling (Wang and Saksena 1999).*

The authors counted two types of context switches (actual and effective) and compared them against those for a fully-preemptive scheduling approach. Effective context switches occur when a thread begins or ends execution, and are important because they directly affect cache performance by disrupting the locality of memory references.

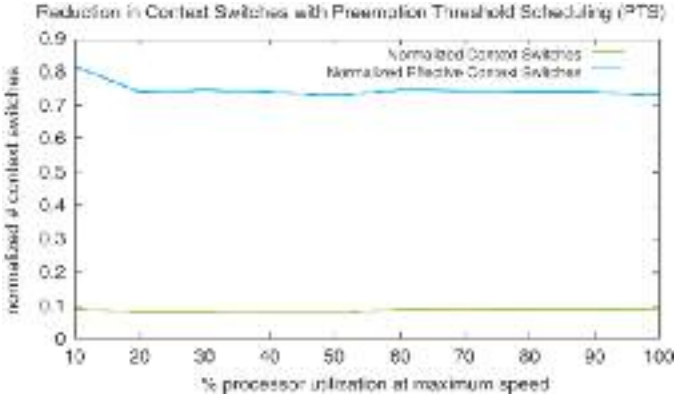


*Figure 2: Number of context switches under PTS compared with preemptive scheduling (Jejurikar and Gupta 2004).*

Jejurikar and Gupta showed that on average, PTS reduced the number of context switches by 90% from the fully-preemptive approach, essentially independent of processor utilization. The number of effective context switches was reduced by 19% to 29%, improving memory system performance by about the same amount.

### PTS Reduces Stack Space Requirements

Threads in a non-preemptive group can share stack space since their execution will not be interleaved. Run-to-completion threads and some blocking threads qualify. Baker's Stack Resource Protocol calculates the possible maximum blocking times of threads, allowing a system designer to determine whether deadlines can be met (Baker 1991). SRP involves modifying the scheduler to not start executing a thread if its preemption level is not high enough. This means that the thread will not block partway through, since it doesn't start running unless all resources are available.

My own research group and others have evaluated how PTS can reduce stack space requirements. PTS can be used with any scheduling algorithm (Rate Monotonic, Earliest Deadline First, etc.)

and it will always render the smallest total stack space requirements (Ghattas and Dean 2007) when using the maximal preemption-threshold assignment algorithm described by Wang & Saksena. Gai extended SRP to support preemption-thresholds and showed that it minimizes total stack space requirements (Gai, Lipari and Natale 2001).

We investigated workload characteristics that affect the stack size reduction achievable through PTS. To cover a wide range of design points, we randomly generated 40,000 systems with 10 threads each. All were schedulable with a fully-preemptive policy.

We first investigated the effect of the system utilization on the stack space required with PTS. The optimal stack space required by each system was computed and normalized to the stack space required by the fully-preemptive version of the system. The space requirements are plotted in Figure 3 as a function of the overall system utilization and the standard deviation in the thread periods ($\sigma_P$).

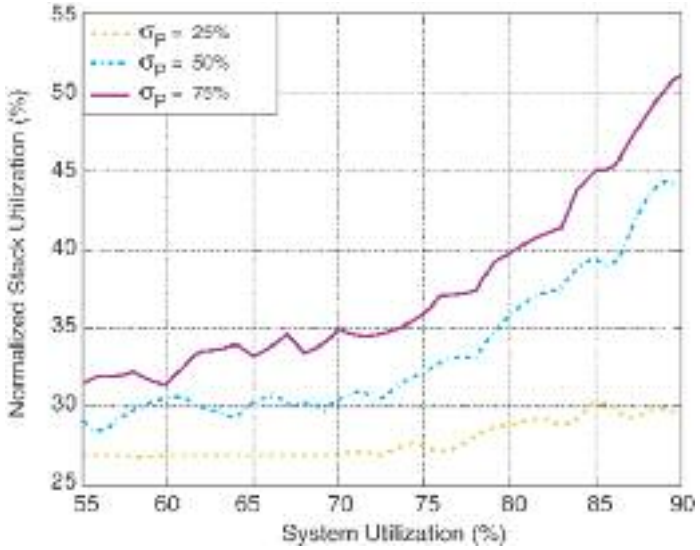


*Figure 3: Stack space required for PTS with a fixed-priority policy rises with both thread period variability and system utilization (Ghattas and Dean 2007).*

Figure 3 shows that at low utilization, the system's stack space requirements might be less than 30% of those of a fully-preemptive system. However, at higher utilization levels, the variation in the threads' periods increasingly affects the savings attainable. With $\sigma_P$ = 25% the savings are only slightly dependent on the utilization level. On the other hand, as the variance and standard deviation of the period increases, the savings attainable decrease at higher utilization. This is because thread execution times are larger so it is much harder to maintain the system's schedulability while minimizing preemptions. This becomes very apparent at high system utilization levels where there is far less slack time.

Figure 4 shows how stack memory requirements are reduced by one half to three quarters by using PTS. This benefit depends on system utilization; the busier the CPU, the more preemptions are needed, so the more stack memory is required. The results for dynamic-priority schemes are similar and are detailed in our paper.

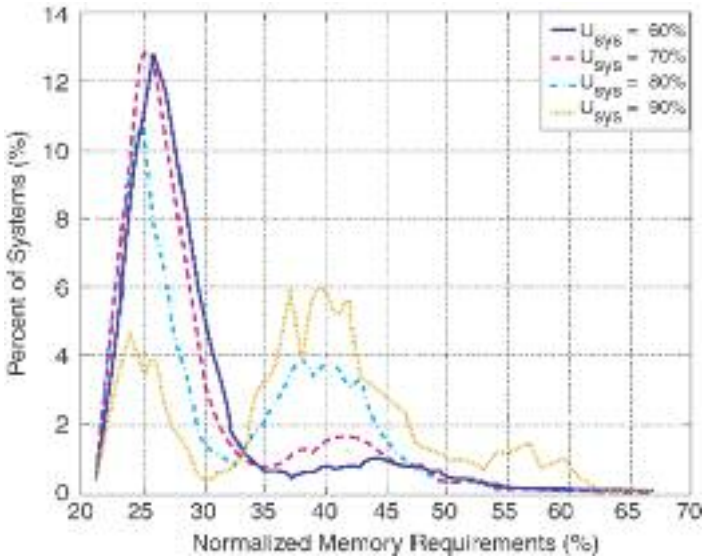


*Figure 4: Stack space required for PTS with a fixed-priority policy rises with both thread period variability and system utilization (Ghattas and Dean 2007).*

## Summary

Express Logic's introduction of Preemption-Threshold Scheduling (PTS) in its ThreadX RTOS enables system designers to reduce preemptions while still meeting real-time deadlines. Reducing preemptions makes even already efficient ARM-based system run faster because less time is expended on context switches. The benefits of such elimination include the ability to increase system operation in low-power mode, and also the reduction in memory needed to meet system stack requirements. There is a well-developed body of theoretical work showing how to use PTS and still meet all system deadlines, making it a no-lose benefit for all ARM-based embedded real-time system requirements

For further information on PTS, and on ThreadX, see: http://rtos.com/products/threadx/.

**END**

**Bibliography**

Baker, Theodore P. "Stack-based scheduling of realtime processes." *Real-Time Systems, 1991.*

Express Logic, Inc. *User Guide: ThreadX: The High-Performance Embedded Kernel.* 2003.

Gai, Paolo, Giuseppe Lipari, and Marco di Natale. "Minimizing Memory Utilization of Real-Time Task Sets in Single and Multi-Processor Systems-on-a-chip." *Proceedings of the 22th IEEE Real-Time Systems Symposium.* IEEE, 2001.

Ghattas, Rony, and Alexander G. Dean. "Preemption Threshold Scheduling: Stack Optimality, Enhancements and Analysis." *Proceedings of the 13th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2007).* IEEE, 2007.

Jejurikar, Ravindra, and Rajesh Gupta. "Integrating Preemption Threshold Scheduling and Dynamic Voltage Scaling for Energy Efficient Real-Time Systems." *Proceedings of the Ninth International Conference on Real-Time Computing Systems and Applications.* 2004.

Wang, Yun, and Manas Saksena. "Fixed priority scheduling with preemption threshold." *Proceedings of the IEEE International Conference on Real-Time Computing.* 1999.

INDUSTRIAL

AEROSPACE


SYSTEM ON A CHIP

MEDICAL

AVIATION

CONSUMER

# THREADX: WHEN IT REALLY COUNTS

**When Your Company's Success, And Your Job, Are On The Line - You Can Count On Express Logic's ThreadX® RTOS**

Express Logic has completed 14 years of successful business operation, and our flagship product, ThreadX, has been used in over 800 million electronic devices and systems, ranging from printers to smartphones, from single-chip SoCs to multiprocessors. Time and time again, when leading manufacturers put their company on the line, when their engineering team chooses an RTOS for their next critical product, they choose ThreadX.

THREADX

Our ThreadX RTOS is rock-solid, thoroughly field-proven, and represents not only the safe choice, but the most cost-effective choice when your company's product simply must succeed. Its royalty-free licensing model helps keep your BOM low, and its proven dependability helps keep your support costs down as well. ThreadX repeatedly tops the time-to-market results reported by embedded developers like you. All the while, Express Logic is there to assist you with enhancements, training, and responsive telephone support.

Join leading organizations like HP, Apple, Marvell, Philips, NASA, and many more who have chosen ThreadX for use in over 800 million of their products – because their products are too important to rely on anything but the best. Rely on ThreadX, when it really counts!

Contact Express Logic to find out more about our ThreadX RTOS, FileX® file system, NetX™ Dual IPv4/IPv6 TCP/IP stack, USBX™ USB Host/Device/OTG stack, and our new PrismX™ graphics toolkit for embedded GUI development. Also ask about our TraceX® real-time event trace and analysis tool, and StackX™, our patent-pending stack size analysis tool that makes stack overflows a thing of the past. And if you're developing safety-critical products for aviation, industrial or medical applications, ask about our new Certification Pack™ for ThreadX.

**expresslogic**

**For a free evaluation copy, visit www.rtos.com • 1-888-THREADX**