

Preemption Threshold

More and more of today's embedded real-time applications use multithreading. The benefits of multithreading are clear and include improved responsiveness, greater throughput, and ease of development and maintenance. However, there are also costs and pitfalls associated with multithreading. The new preemption-threshold technology found in ThreadX helps eliminate some of these pitfalls.

MULTITHREADING

What is multithreading? Multithreading is simply a technique used for processor allocation. Applications are divided into pieces, commonly called threads, and a kernel like ThreadX coordinates their execution. Of course, not all threads are created equally. Some threads of an application have greater importance or priority than others. In ThreadX, each application thread has an assigned priority. ThreadX executes the highest priority, ready-to-execute thread first. Furthermore, the execution of a lower-priority thread is preempted when one of more importance becomes ready. This style of multithreading is the most common and is generally referred to as preemptive, priority-based scheduling.

Preemptive, priority-based scheduling is very powerful, but there are several pitfalls to watch out for that can affect system responsiveness and overall system throughput.

ThreadX provides real-time applications with a new technology called preemption-threshold which allows a thread to only disable preemption of threads up to a specified threshold priority.

application that has interrupts locked out for long periods of time! Another solution is to disable thread preemption. This type of capability is found in most multithreading run-time environments. Disabling preemption allows interrupts but does not allow any other threads — regardless of their priority — to execute. Of course, this is overkill for many critical sections since protection is only needed from a group of thread priorities — not all of them!

ThreadX provides real-time applications with a new technology called preemption-threshold. Preemption-threshold allows a thread to only disable preemption of threads up to a specified threshold priority. Threads having priorities higher than the threshold are still allowed to preempt. By utilizing preemption threshold, high-priority threads can continue to respond — in real-time — to the external world and not be blocked by unrelated critical section processing of lower-priority threads.

SYSTEM RESPONSIVENESS

Most applications have situations where threads must perform certain actions with protection from other high-priority threads. Examples of this include manipulation of shared system resources and control of peripheral hardware devices. The areas of threads where these situations occur are commonly called critical sections.

Most multithreading run-time environments offer two solutions to this thread protection requirement. The first solution is to disable interrupts. Doing this certainly keeps other higher-priority threads from preempting, but at a high cost. Nothing like a real-time

Another unpleasant pitfall associated with priority-based, preemptive scheduling is priority inversion. Priority inversion takes place when a higher-priority thread is suspended because a lower-priority thread has a needed resource. Of course, in some instances it is necessary for two threads of different priorities to share a common resource. If these are the only threads active, the priority inversion time is bounded by the time the lower-priority thread holds the resource. This condition is both deterministic and quite normal. However, if a thread of an intermediate priority becomes active during this situation, the priority inversion time is no longer deterministic and could result in total system failure. Preemption-

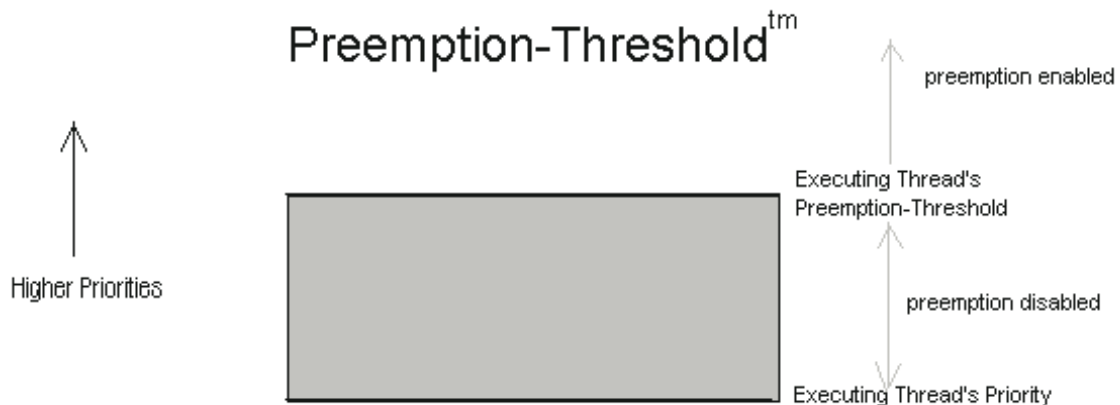


Figure 1. Preemption Threshold

threshold technology helps application developers handle the priority inversion problem. By disabling preemption between the executing lower-priority thread and the higher-priority thread, priority inversion is now guaranteed to be bounded by the time the lower-priority thread holds the shared resource. The higher-priority thread is still allowed to preempt and perform as much processing as possible until the shared resource is needed.

SYSTEM THROUGHPUT

Another pitfall associated with priority-based, preemptive multithreading is reduced system throughput. Of course, the most obvious method to improve system throughput in multithreading is to reduce the number of context switches. As previously mentioned, ThreadX performs a context switch when a higher-priority thread becomes ready during the execution of a lower-priority thread. It is important to mention that this condition occurs as a result of both interrupt-driven events as well as ThreadX services called from within a thread.

To illustrate the effects thread priorities have on context switch overhead, assume three threads with names thread1, thread2, and thread3. Assume further that all threads are in a state of suspension waiting for a message. When thread1 receives a message, it immediately forwards it to thread2. Thread2 then forwards the message to thread3, which then discards the message. After each thread processes their message, they re-suspend waiting for another message.

The processing required to execute these three threads varies greatly depending on their priorities. If all of the threads have the same priority, a single context switch occurs between their execution — at the point where each thread suspends waiting for another message. If, however, thread2 is higher-priority than thread1, and thread3 is higher priority than thread2, the number of context switches doubles. An extra con-

text switches occur inside of ThreadX's queue send service because each queue send operation results in a preemption condition.

Applications can utilize preemption-threshold to eliminate these excessive context switches and still allow the flexibility of having different priorities. This is accomplished by insuring the preemption-threshold of threads1 and threads2 disables preemption of priorities between themselves and that of thread3. For example, assume thread1 has a priority of 30, thread2 has a priority of 29, and thread3 has a priority of 28. If the preemption-thresholds of thread1 and thread2 are set to 28, they are not preempted by any threads with priorities between 30 and 28. Of course, this eliminates the extra context switches associated with these different priority threads.

CONCLUSION

Preemption-threshold is an important new technology that helps improve real-time responsiveness as well as reduce excessive context switching overhead. Best of all, ThreadX provides preemption-threshold with virtually no additional overhead when compared with other commercial kernels — basically an integer compare instead of checking a preemption-enable flag.

Preemption-threshold is a significant new technology and is representative of the other advancements ThreadX brings to embedded software development.

Bill Lamie has 15 years of experience as an embedded software engineer. He is the principal architect of several commercial kernels, including ThreadX, the high-performance kernel offered by Express Logic, Inc. Bill may be contacted at blamie@expresslogic.com or by phone at (619) 674 6684.