Microsoft

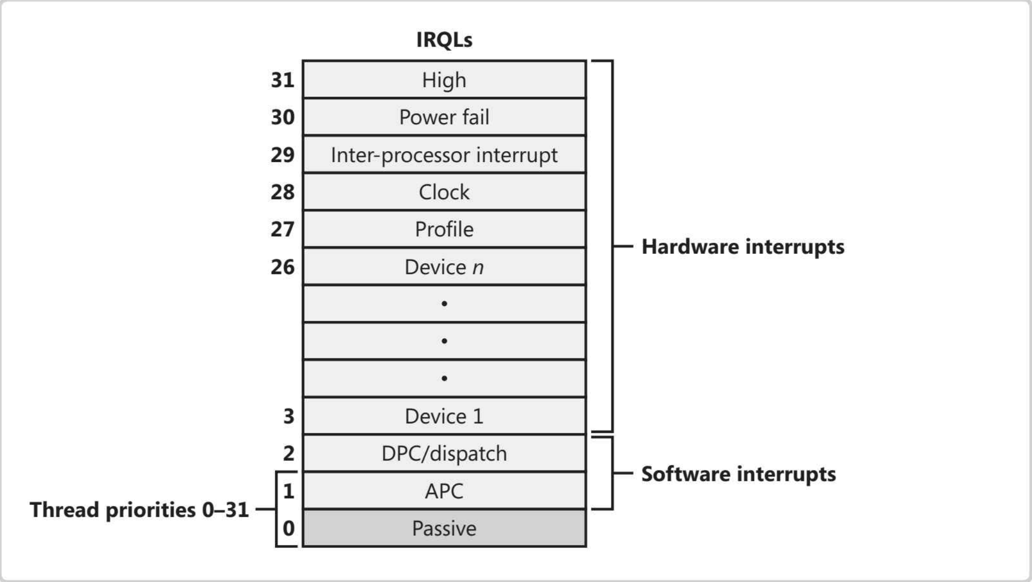
Foreword by Ben Fathi
Corporate Vice President, Windows Core Development, Microsoft Corporation

5
FIFTH EDITION

Windows® Internals

Covering Windows Server® 2008 and Windows Vista®

Mark E. Russinovich
and David A. Solomon
with Alex Ionescu



Thread States

Before you can comprehend the thread-scheduling algorithms, you need to understand the various execution states that a thread can be in. Figure 5-14 illustrates the state transitions for threads. (The numeric values shown represent the value of the thread state performance counter.) More details on what happens at each transition are included later in this section.

The thread states are as follows:

- **Ready** A thread in the ready state is waiting to execute. When looking for a thread to execute, the dispatcher considers only the pool of threads in the ready state.
- **Deferred ready** This state is used for threads that have been selected to run on a specific processor but have not yet been scheduled. This state exists so that the kernel can minimize the amount of time the systemwide lock on the scheduling database is held.
- **Standby** A thread in the standby state has been selected to run next on a particular processor. When the correct conditions exist, the dispatcher performs a context switch to this thread. Only one thread can be in the standby state for each processor on the system. Note that a thread can be preempted out of the standby state before it ever executes (if, for example, a higher priority thread becomes runnable before the standby thread begins execution).
- **Running** Once the dispatcher performs a context switch to a thread, the thread enters the running state and executes. The thread's execution continues until its quantum ends (and another thread at the same priority is ready to run), it is preempted by a higher priority thread, it terminates, it yields execution, or it voluntarily enters the wait state.

- **Waiting** A thread can enter the wait state in several ways: a thread can voluntarily wait for an object to synchronize its execution, the operating system can wait on the thread's behalf (such as to resolve a paging I/O), or an environment subsystem can direct the thread to suspend itself. When the thread's wait ends, depending on the priority, the thread either begins running immediately or is moved back to the ready state.
- **Gate Waiting** When a thread does a wait on a gate dispatcher object (see Chapter 3 for more information on gates), it enters the gate waiting state instead of the waiting state. This difference is important when breaking a thread's wait as the result of an APC. Because gates don't use the dispatcher lock, but a per-object lock, the kernel needs to perform some unique locking operations when breaking the wait of a thread waiting on a gate and a way to differentiate this from a normal wait.
- **Transition** A thread enters the transition state if it is ready for execution but its kernel stack is paged out of memory. Once its kernel stack is brought back into memory, the thread enters the ready state.
- **Terminated** When a thread finishes executing, it enters the terminated state. Once the thread is terminated, the executive thread block (the data structure in nonpaged pool that describes the thread) might or might not be deallocated. (The object manager sets policy regarding when to delete the object.)
- **Initialized** This state is used internally while a thread is being created.

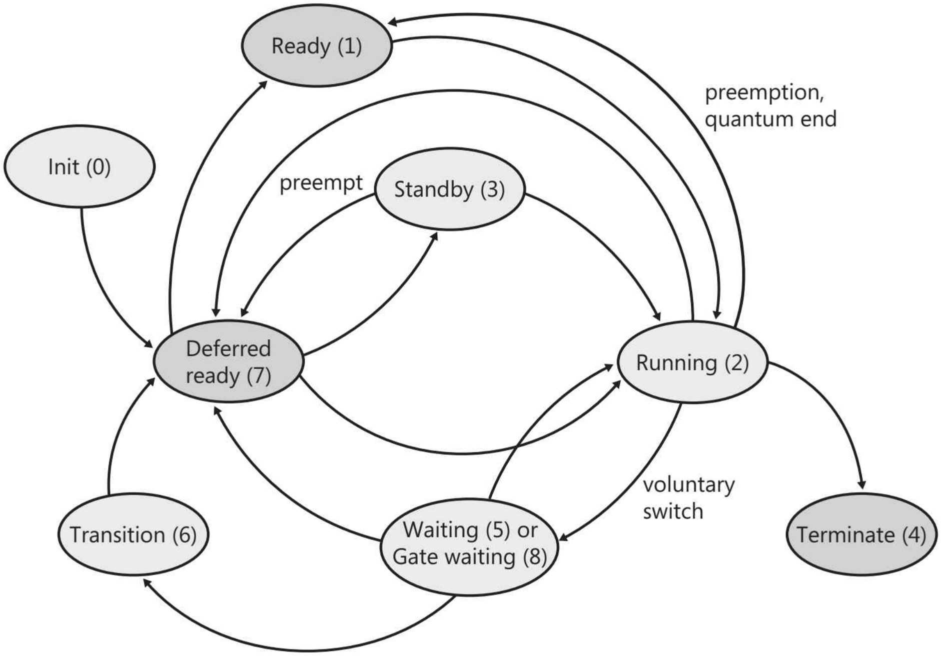


FIGURE 5-14 Thread states and transitions