The 4D v11 SQL Scheduler

By Josh Fletcher, Technical Services Team Member, 4D Inc.

Technical Note 09-45

Table of Contents

Abstract

As with previous versions of 4D, 4D v11 SQL uses an internal scheduler to manage 4D processes. To some extent the Scheduler has always been a bit of a black box. Though it can be tweaked, it may not be fully understood what the changes affect.

This Technical Note invites the reader to explore the Scheduler in detail to gain a better understanding of how 4D works under the hood and to understand how the Scheduler can be tweaked.

Introduction

The data engine for 4D v11 SQL was rewritten from the ground up to be multi-threaded in order to take advantage of the latest multi-core and multi-processor machines. However the fact remains that the 4D Language is not a thread safe language. Thus all 4D code still executes in a single cooperative thread.

As in the past, 4D v11 SQL employs the time-tested Scheduler to allow 4D processes to multi-task. Of course the Scheduler has never been static and, even in 4D v11 SQL, tweaks have been made to strike a good compromise between performance and scalability.

This Technical Note presents a picture of how the Scheduler works with the goal of empowering the 4D developer to further understand how to squeeze every last bit of performance out of their applications.

Scheduler Overview

At the highest level, the Scheduler is simply a loop that continues forever until the application quits. It has two primary tasks:

- Check for events from the system and pass those on to 4D processes.
- Give some time to all 4D processes to perform their tasks.

In psuedocode, the Scheduler looks like this:

```
While 4D is running

   // Process system events.
   Repeat
      If ticks_between has passed
         If 4D is busy
               timeout = min_ticks
         Else
               timeout = max_ticks
         End if
```

```
         Get the next event or wait for timeout // This yields to the system.
         If there is an event for a 4D process
               Pass the event to the waiting process
         End if
      End if
   Until there are no more events

   // Give some time to each 4D process.
   For each 4D process
      Give at least 1 tick to each active process

End while
```

The three bold items correspond the values that the developer can actually tweak. These can be set with the SET DATABASE PARAMETER command (explained in the next section).

Some interesting points to note:

- If **ticks_between** has not passed since the last system call to check for an event, 4D simply keeps giving time to 4D processes (the Repeat loop drops through and the For loop executes again). In other words 4D does not yield control to the system and instead keeps using CPU time.
- 4D keeps track of how busy it is based on what the 4D processes are doing. Processes that are delayed, paused, or waiting for semaphores are considered "not busy". Processes that are executing are, of course, considered "busy". While the full details of how 4D determines whether or not it is busy cannot be revealed, if a common sense look is taken at the possible states for a 4D process, a decent picture of how it works can be seen.
  - The point being, if 4D is busy, it uses more CPU time because it yields to the system for less time. If 4D is not busy it yields for a longer time period. (By default).
- When 4D is giving time to processes (the For loop) only "active" processes consume time. Processes that are delayed, paused, or waiting for semaphores do not consume time.
- Active processes consume at least 1 tick (even if they do not need all of it). This means that the minimum amount execution time for the For loop is *number of active processes x 1 tick*.
  - Keep this in mind when building any system that relies heavily on "pooling". Be sure to pause or delay "pooled" processes so that they do not adversely affect performance by consuming extra CPU time.

Please note none of this applies to the pre-emptive portions of 4D. While there are certainly cases where pre-emptive code needs to be synchronized with 4D processes, for the most part the 4D Scheduler only manages cooperative 4D code.

Also note that though this document is here to provide a deeper understanding of 4D, the information provided here is subject to change at any time.

Finally, keep in mind that there is absolutely no reason a developer **must** tweak the scheduler. For most situations the default values are fine. The goal of this Technical Note is to help the developer investigate those scenarios where the default settings may not be optional and, through deeper understanding, be able to make educated decisions about what changes to make.

Tweaking the Scheduler – SET DATABASE PARAMETER

The Scheduler has 3 possible user-editable values as shown in the previous psuedocode:

- min_ticks: Minimum time per call to system
- max_ticks: Maximum time per call to system
- ticks_between: Ticks between calls to system

These values can be set for each type of 4D application (Server, Remote, Local) but this distinction is not important for understanding what the values do; they do the same thing for each application type.

Recall that 1 tick is 1/60th of a second.

These values are discussed in the following sections.

Min Ticks

This value is used to determine how long 4D yields control of the CPU to the system when 4D is "busy". It is intended to allow the system potentially less CPU time, per call to the system, when 4D is busy. Note that this is not always the case. When 4D yields to the system it either gets the next event (which returns control immediately) or waits for the indicated timeout period. Events may arrive during this period and shorten the duration of the wait.

Also note that **4D does not decide when it gets control**. It is up to the OS to return control to 4D.

Increasing this value means that when 4D is busy it will yield control to the system for longer periods of time and thus potentially increase the interval between giving some processing time to 4D processes (i.e. there is more time between each For loop execution in the psuedocode).

Decreasing this value means that when 4D is busy it will yield control to the system for shorter periods of time. Therefore 4D processes will have access to CPU more often (the For loop will execute more often).

The default value is 0. The range is 0 to 100.

Max Ticks

This value is used to determine how long 4D yields control of the CPU to the system when 4D is "not busy". It is intended to allow the system to get the CPU for more time, per call to the system, when 4D is not busy. Note that this is not always the case. When 4D yields to the system it either gets the next event (which returns control immediately) or waits for the indicated timeout period. Events may arrive during this period and shorten the duration of the wait.

Increasing this value causes 4D to consume less CPU time when it is idle. This can be especially beneficial in environments where the machine needs to run more than one application and/or on machines with a limited number of CPUs/cores.

Decreasing this value is an easy way to cause processor "spoofing" (see the Processor Spoofing section below) but also ensures that 4D has more CPU time and, therefore, spend more time running 4D code.

The default value is 8. The range is 0 to 100.

Ticks Between

This value determines how long 4D keeps control of the CPU between calls to the system. The higher this value is, the more time 4D spends looping through its own code before it yields to the system (no matter how busy it is).

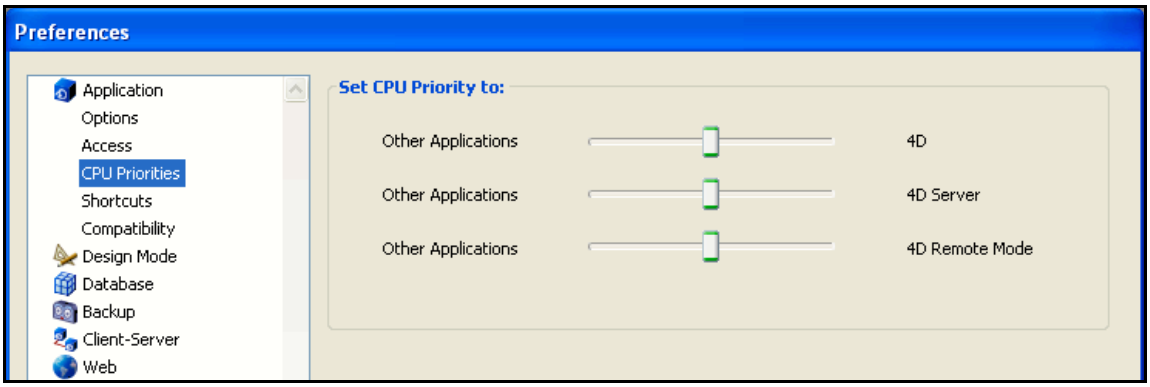
Increasing this value ensures more CPU time for 4D processes (essentially allowing the For loop in the psuedocode to execute more often). This can also lead to processor spoofing if 4D is idle.

Decreasing this value causes 4D to yield to the system more often (essentially allowing the Repeat loop in the psuedocode to execute more often).

The default value is 0. The range is 0 to 20.

Tweaking the Scheduler – Preferences

Of course the Scheduler settings can be adjusted in the database Preferences, under Application | CPU Priorities:



However to some extent, without the Scheduler Psuedocode and an understanding of what SET DATABASE PARAMETER does, these settings are a bit of an enigma. Let's look at the actual values that these settings produce (this can be checked with GET DATABASE PARAMETER), armed with a deeper understanding of what they mean:

Setting	Min Ticks	Max Ticks	Ticks Between
Max to 4D	0	1	5
Average to 4D	0	8	0
Max to Other Apps	1	16	0

Average to 4D

This is the default setting. When 4D is busy it yields to the system for 0 ticks. When 4D is idle it yields to the system for up to 8 ticks. 4D does not hang on to the CPU between calls to the system for any minimum amount of time. Note that this does not mean no CPU time passes between calls to the system. This would, of course, be impossible. It just means that 4D doesn't guarantee a minimum amount of execution time for 4D processes.

Max to 4D

With this setting, 4D's behavior when busy is unchanged. However when 4D is idle it still only yields to the system for 1 tick. Finally 4D hangs on to the processor for at least 5 ticks between calls to the system. The net result is that 4D consumes more processor time than the default setting.

Max to Other Apps

With this setting, 4D is the most "passive". When busy it still yields to the OS for up to 1 tick. When idle it yields for up to 16 ticks. Finally 4D makes no attempt to hang on to the processor between calls to the system. The net result is that 4D consumes less processor time than the default setting.

Processor Usage

So what does it mean if 4D uses less CPU time? The simple answer is it depends.

If 4D is largely idle (meaning 4D processes are mostly paused, delayed, waiting for semaphores, etc.) it might not mean much of anything if 4D uses less CPU. Perhaps the only major problem with this scenario is that 4D will be less responsive when processes need to wake up; by yielding to the system for longer periods of time, those waiting processes aren't being notified as often.

On the other hand if intensive tasks are being performed, like looping over a selection or an array, the more CPU time those 4D processes have, the faster they will complete.

It really is a matter of striking the right balance between completing 4D tasks as quickly as possible and allowing the system to complete its own tasks (including running other applications).

Multi-Processing

It is important to note that in modern computers, which typically have access to more than one CPU/core, it is less of a problem to dedicate more CPU time to the 4D Scheduler (and thereby 4D processes). Remember that 4D code is executing in a single cooperative thread; it will only ever monopolize 1 CPU/core. Depending on the design of the database it may be advantageous to give more CPU time to 4D code.

Keep in mind that there are other, non-parallel bottlenecks though. Network access, for example, will be a bottleneck if there are several applications vying for it.

Also note that the Scheduler settings can be changed on the fly (via SET DATABASE PARAMETER) so it is possible to tune 4D for specific scenarios like increasing the CPU time before kicking off an intense report process and then setting it back when done.

Processor Spoofing

Processor spoofing is a phenomenon that occurs when 4D requests CPU time from the system but yields it back nearly immediately because it is idle. By that same token, the system may immediately return control to 4D because it has nothing else to do. The net result is that 4D appears to be using a lot of CPU time, even though it isn't doing anything.

But, in fact, 4D is doing "something"…it is looping in the Scheduler. This is something that makes 4D unique among other applications. Most applications are passive, awaiting user input. Multi-threaded applications are passive in the

sense that the OS decides when the threads get to run. But 4D's scheduler is actively running, all the time. In fact 4D shares this idea in common with the OS. The OS is never idle and in fact the CPU usage is never "0". Computers are clocked devices…if the computer is on, the electricity is flowing and the CPU is being used. On Windows the "System Idle Process" occupies unused CPU cycles and on Mac it is reported as the "IDLE" percentage. In fact these reporting tools are really showing you the difference in CPU usage between competing threads. Another way to say it: the CPU is always at 100% (ignoring energy saving technologies that lower the clock rate). It's just a matter of who's using the cycles.

The 4D Scheduler is always running in a loop. It has two "modes" of running, by default.

- If 4D thinks it's busy, based on the tasks it is doing, it yields to the OS only momentarily and request control again (min_ticks).
- If 4D thinks it is not busy, it yields to the OS for 8 ticks (by default, max_ticks)

In the first "mode" the processor usage will be reported higher.

In the second "mode" 4D will appear more idle.

The most important thing to understand is that, either case, **this is not a direct correlation to performance!** That is to say, the proper way to measure performance is not by watching the CPU usage. The CPU usage is only a snapshot, taken at certain intervals. Imagine a scenario where every time the Task Manager polls the CPU it just so happens 4D has control. This will "spoof" the CPU usage even though 4D might be just about to hand control back to the OS.

The proper way to measure performance is **to measure performance!** Measure how long things take to complete. Are other applications still responsive? Is 4D completing tasks in acceptable time? These are the metrics that should be used to measure performance, not CPU usage.

Also, because 4D is always requesting CPU time in a loop, if the machine itself is mostly idle, then the OS will give the time to 4D more often. This also causes the "spoofing" scenario.

Note: *An abnormal situation can occur when the preferences for the Scheduler are corrupt. If the CPU usage for 4D appears abnormally high and the Scheduler settings are normal, try toggling the settings (set it to High, close the Preferences, open them back up, set it to Normal) and see if that fixes it.*

Conclusion

The Scheduler is a tried and tested feature of 4D. Though 4D v11 SQL was rewritten to take advantage of modern CPU architectures, the Scheduler remains in order to ensure compatibility with the 4D language.

Tweaking the Scheduler has been a bit of a mystery over the years. This Technical Note shined a new light on this important aspect of 4D's architecture in order to empower the 4D developer to take control of this feature.