Technical Note 05-20
CPU、スケジューラ、プロセス

By Jean-Yves Fock-Hoon, QA Manager
Technical Note 05-20

（原題: CPU, scheduler and processes）

概要

4D の内部構造に関する基本的な知識は、データベースの速度向上に役立ちます。

CPU 優先順位

実行中の各プロセスには優先順位が存在し、それぞれがシステムにより配分された CPU 時間を利用しています。

4D の環境設定には CPU の優先順位に関する項目があり、低、通常、高の 3 通りの設定が可能です。もっと細かいカスタム設定をするには SET DATABASE PARAMETER コマンド（10、11、12 番）を使用します。

セレクタ	定数	適用
10	4th Dimension Scheduler	4th Dimenison
11	4D Server Scheduler	4D Server
12	4D Client Scheduler	4D Client

このコマンドはいつでも使用することができます。
現在値を取得するには次のようにします。

```
$lparams:=Get database parameter(4th Dimension Scheduler )
vNbTicks:=$lparams & 0x00FF
vMaxTicks:=($lparams >> 8) & 0x00FF
vMinTicks:=($lparams >> 16) & 0x00FF
```

続けて次のように設定を変更することができます。

```
$lparams:=vNbTicks+(vMaxTicks << 8)+(vMinTicks << 16)
SET DATABASE PARAMETER(4th Dimension Scheduler ;$lparams)
```

vMinTicks および vMaxTicks は、システムコールをする際に確保される時間の最小・最大時間です。システムコールが長ければ、それだけ長く CPU 時間は OS の管理下に置かれ、他のアプリケーションに配分されるか、どのアプリケーションにも使用されず無駄になる可能性が高くなります。

この値が低いことは、より多くの CPU 時間が 4D に割り当てられることを意味しますが、これは OS や他のアプリケーションを遅くする原因になります。4D Client の場合、OS の時間が奪われる結果、接続切断の回数が増すことにもなりかねません。ネットワークが遅い場合は、CPU に多くの時間を当てることによって、パフォーマンスを改善できることがあります。

最小最大の幅は大きければ、4D が可能なときに CPU を占有し、システムが必要とするときには譲ることを意味します。4D は速くなりますが、他のアプリケーションが多くの CPU 時間を要求すると遅くなります。

NbTicks は、4D 自身の CPU 時間を解放するまでの待ち時間です。

通常はデフォルトの設定で十分なはずですが、もし設定を変更するようなときには、他のアプリケーションやシステムとのバランスを考慮することが肝心です。

スケジューラ

4D は、自らがスケジューラを持つスレッドなので、OS から CPU 時間が与えられると、その一部は 4D 内の各プロセスに配分するため、スケジューラによって消費されます。スケジューラは、他のアプリケーションのために適当なときに CPU 時間を解放します。問題は、ある 4D プロセスが他と比べてより多くの CPU 時間を要し、他の 4D プロセスに十分の時間が行き渡らない場合です。

クライント/サーバ環境では接続が切れるかも知れず、もっと深刻な状況として、スケジューラに制御が戻らない結果、システムおよび他のアプリケーションまでが CPU 時間の欠乏に見舞われる可能性があります。

インタプリタモードで実行中のときには、コマンドやメソッドを実行する際に必ずスケジューラに制御が渡りますが（コールバック）、コンパイルして単一のプロセスを実行し、まったくコールバックをしなければ、プロセスがずっと CPU 時間を占有する危険があります。

4D は、主立ったコマンドを実行する際には、必ずコールバックをするようになっていますが、最適化をはかるため、ある種の簡単なコマンドはいちいちコールバックを実行しません。これにはループ（For 文、While 文 etc.）も含まれます。

```
For ($i;1;200000)
    $MyVar:=4
End for
```

この記述は、コンパイルモードではコールバックを実行しません。したがって IDLE をはさむことによって、スケジューラに制御を戻すことが望ましいとされています。そうすれば、スクジューラは CPU 時間を適切に処理することができます。

アイドリング

主立ったコマンドはコールバックをしますが、そうしないものもあるので、場合によってはIDLEを実行する必要があります。実のところ、IDLEコマンドそのものがコールバックを実行するのではありません。IDLEはコールバックを要求するのであり、このときスケジューラは各プロセスが最低1ティックを消費するようにします。仮にIDLEを10回連続で実行するメソッドがあったとしても、この処理自体が1ティック内で終了すれば意味がありません。1ティックが消費されるまで、スケジューラとプロセスの間で制御が往復するだけです。確実にコールバックを実行するにはDELAY PROCESS(0)を使用します。

マルチプロセスでは、各プロセスに適度なCPU時間が配分されるように気を配ることが肝心です。とはいえIDLEやDELAY PROCESSをあまりに多用すると、かえってパフォーマンスが落ちる恐れがあるので、その点も注意が必要です。

マルチプロセス

通常のユーザは、使用しないプロセスを消す代わりに隠すだけにします。CPUにとってこれは何の相違もなく、依然としてスケジューラはそのプロセスに時間を配分します。実際、隠されたプロセスについてはPAUSE PROCESSを実行するべきです。あるいはDELAY PROCESSを実行することもできます。DELAY PROCESSを実行すれば、一定期間、スケジューラはそのプロセスにCPU時間を配分しなくなります。

頻繁にプロセスが必要になったり不要になったりする場合には、プロセスプールを使用する方法が有効です。使用しないプロセスはPAUSEしてプールに移動し、再び必要になったときに再開するようにします。プロセスプールが空のときは新規プロセスを立ち上げます。4D Client自体、このような手法を用いており、Web Serverでそのようにしている4Dデベロッパもいます。

新規プロセスを立ち上げると、4Dはそのプロセスの変数用にメモリを確保し、プロセス情報を更新します。変数のリストが膨大な場合や、遅いクライアント/サーバにおいてこの処理は非常にCPUへの負担が大きくなります。毎回、削除と作成を繰り返すことを考えれば、プールの発想は非常に合理的です。

4D Clientと4D Serverの場合は別の問題があります。4D Clientと4D Serverはそれぞれがスケジューラを持っており、通信に関わるプロセスにCPU時間を割り当てています。どちらか一方の側で十分の時間がプロセスに行き渡らないと、内部タイムアウトエラーが発生してこれらのプロセスが消えてしまうかもしれません。さらに他方のスケジューラが通信に関わるプロセスにCPU時間を割り当てる頃には、相手が消えているので、ここでもタイムアウトエラーが発生することになります。

クライアントがあまりに忙しすぎると、4D Serverに反応できなくなり、4D Serverのクライアントリストから消えてゆくことになります。4D Clientがついに通信を回復する頃には、もう手遅れてネットワークエラーが発生します。

サーバがあまりに忙しすぎると、4D Clientに反応できなくなり、4D Clientはそのサーバが閉じたものと判断して、エラー10002メッセージを出力します。

このように、たとえサーバであってもあまりに多くのプロセスを同時に立ち上げることは避けたいものです。データアクセスを必要としないプロセスの場合は、ローカルプロセスで実行するというのも実際的な方法です。

その他の注意点

プロセスプールについては前述しましたが、このプール自体が多くのメモリを消費するものなので、変数テーブルを最適化し、オブジェクトの数を少なく抑えるなど、メモリを効率よく使用するような工夫が求められます。

フィールドにインデックスがはられている場合、各プロセスがレコードを更新、削除するたびにインデックステーブルも更新されるので、無闇にインデックスをはるのは避けるべきです。

さらにトリガ、イベントなどは、使用するものだけを有効にし、ボトルネックになるような複雑なトリガは避けるようにして下さい。複雑な処理は、プロジェクトメソッドやフォームメソッドで実行し、トリガは直接データの保存や読み込みに関係する最低限のデータベースメソッドだけを記述するようにします。トリガ以外のメソッドのほうが、他のプロセスに与える影響が少ないからです。