4D v11 SQL の新しいマルチスレッドモデル

By Atanas Atanassov, Technical Services Team Member, 4D Inc.

Technical Note 10-06

目次

概要

アプリケーションを構築するプロセスにおいて、4D のデベロッパーは自らのアプリケーションのビジネスロジックを完全にコントロールすることができます。デベロッパーは 4D の新しいマルチスレッドモデルを利用することができます。他方、正しいプログラミングを行わなければ利用可能なシステムリソースを使用することができず、システムパフォーマンスが低下し、遅くてときには応答がないようなアプリケーションとなってしまうかもしれません。そこで 4D の新しいスレッドモデルと、異なるタスクがマルチコアアーキテクチャーでどのように動作するのかを正しく理解することが重要です。

はじめに

新しいスレッドモデルにより、4D はマルチコアアーキテクチャーを使用できるようになりました。すべてのプロセスは異なるシステムスレッドで実行されるので、オペレーティングシステムはその実行を複数のコアに振り分けることができ、利用可能なシステムリソースを使用することができます。UI ダイアログプロセスやデータ入力などでのユーザーI/O 待ち、あるいは他のプロセスによるブロックなどの理由で、オペレーティングシステムはプロセスを一時停止させることがあります。インデックス、データフラッシュ、バックアップなどの他のプロセスはバックグランドで走ることができます。これらのプロセスはユーザー入力待ちが無いため、実行をマルチコア間で分割し、よりよいパフォーマンスを得ることが可能です。

新しいマルチスレッドモデルにより、全体のパフォーマンスを低下させることなしに、複数のアプリケーションをひとつに結合させることができます。例えば 4D Server v11 SQL はアプリケーションサーバー、Web サーバー、そして SQL サーバーを含んでいます。これらすべてのアプリケーションは同時に異なるスレッドで走ることができ (この点についてはこのテクニカルノートで後述します)、ユーザーに対しては完全に透過的です。これらのアプリケーションへのすべてのリクエストは異なるポートからやってきます。ポート番号 19813 あるいは Web サーバーへのリクエストはコオペラティブスレッドで実行され、ポート 19812 と 19814 へのリクエストはプリエンプティブスレッドで実行されます。

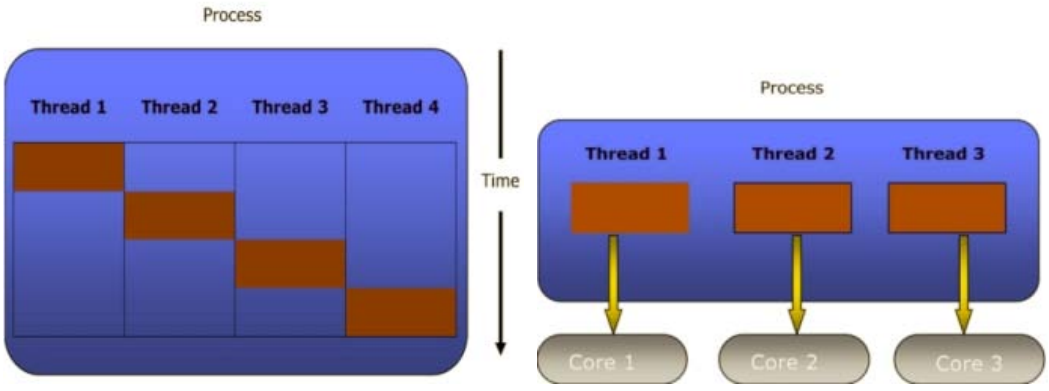
このテクニカルノートでは、どのように実装されどうやって利用するのかなど、4D の新しいマルチスレッドモデルを説明します。最初の部分ではマルチスレッドの一般的な知識を得るために、コオペラティブスレッドとプリエンプティブスレッドについて、そしてこれらがシングルコアやマルチコアのシステム上でどのように実行されるのかを説明します。次の部分では 4D におけるマルチスレッドで、異なるタスクが異なるスレッド上でどのように実行されるのか、そして新しいモデルから得られる全体的な利点を説明します。

マルチスレッドを理解する

スレッドとはアプリケーションの実行単位です。実行は同時に走る 2 つ以上のタスクやスレッドに分割されます。スレッドはプロセスの内側に含まれます。同じプロセス内に複数のスレッドが存在でき、メモリーなどのリソースを共有できます。異なるプロセスはこれらのリソースを共有できません。

スレッドを構成するものには以下があります:

- メモリースタック: メモリースタックはメモリー中の予約された場所で、格納されたデータは先入れ先出しで管理されます。ここには、戻り値が正しい場所に返されるようにするため、関数呼び出しの場所が保持されます。メモリー量は関数が存在するときに、オペレーティングシステムから自動で再要求されます。このため、スタックベースのメモリー割り当ては一時的なデータの格納にとても便利です。スレッドのスタックサイズは通常小さなものですが、アプリケーションの実行コンテキストは複数のタスクから構成されるので、スレッドのスタックにより多くのメモリーを割り当てると、スタックオーバーフローによるクラッシュの原因となることがあります。
- CPU レジスターの状態: CPU レジスターは CPU 上で利用できる少量のストレージです。データは RAM (Random Access Memory) から転送され、実行命令や CPU 命令セットからの計算により処理されます。これにより CPU は非常に高速にデータにアクセスでき、データを処理して、送り出すことができます。
- システムスケジューラー中の実行リストのエントリー: システムスケジューラーはタスクの実行を制御します。この制御はタイムスライスまたはマルチコアのスレッドです。Windows XP 以降および Mac OS X のような新しいオペレーティングシステムはすべて、両タイプのスレッドをサポートしています。以下はタイムスライス (左側) とマルチコアスレッド (右側) の例です。



タイムスライスされたスレッドはプロセスが 1 つのコアで実行されている場合に使用されます。通常シングルコアマシンまたはスレッド実行の結果が共有リソースに影響する場合です。

マルチコアスレッドはマルチコアアーキテクチャー上でスレッド実行がプロセスの共有リソースに影響しない場合に使用されます。

マルチスレッドアプリケーションは同時に複数のスレッドを作成し実行する能力を持ったアプリケーションです。これらのスレッドをシングルコア上で走らせるために、オペレーティングシステムはまるで同時に実行されているかのように、CPU タイムをスレッド間で分割します。このリソース共有は CPU 時間のほとんどがアイドルであるがゆえに可能です。例えばユーザーがデータを入力中、システムがビジーであるかのような印象がありますが、実際にはこのプロセスの CPU 利用率は 0%です。マルチコアマシンの場合、オペレーティングシステムはスレッドの実行をすべてのコア間で、稼働率に基づいて分割します。

実行コンテキストに基づくアプリケーションスレッドは 2 タイプのスレッド、コオペラティブとプリエンプティブに分けられます。

コオペラティブスレッド

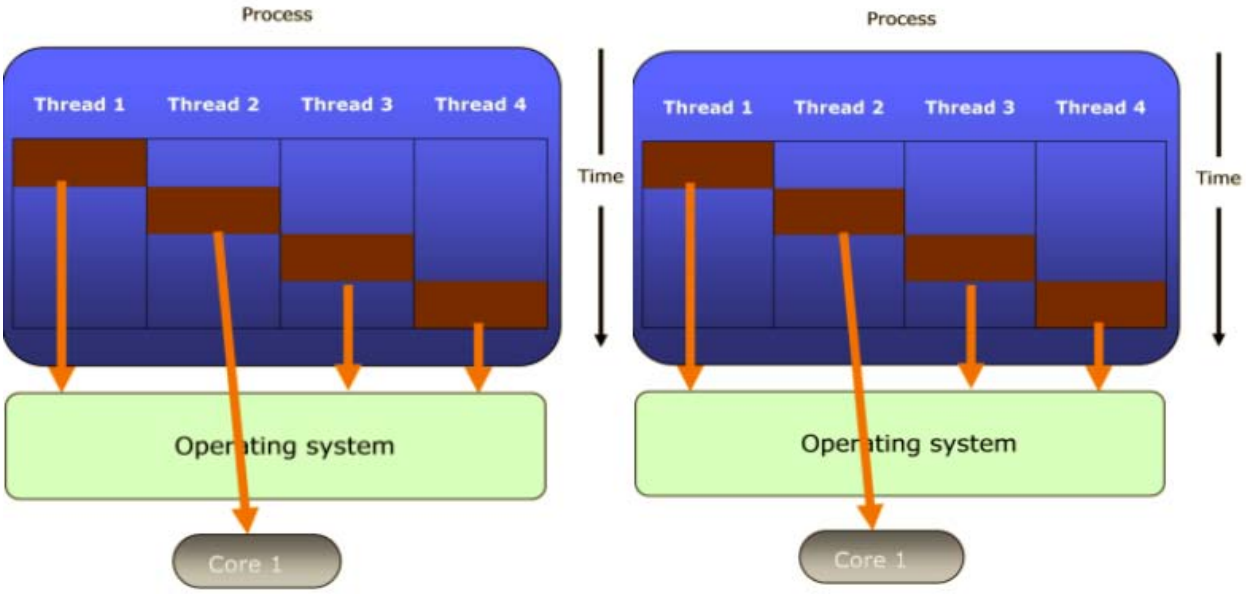
マルチコアアーキテクチャーにおいてコオペラティブスレッドは、1 つのスレッドの実行結果が共有プロセスリソースに影響するため、常に 1 つのコア上で実行されます。異なるスレッドからの共有リソースに対する同時の更新や読み込みは予期しない結果を生むことがあり、デッドロックを起こす可能性があります。

コオペラティブスレッドはユーザーの I/O のために一時停止されるまで、実行します。実行はデータを取得すると再開されます。待ちの間、アプリケーションは制御をオペレーティングシステムに返します。オペレーティングシステムは走る準備ができている他のスレッドに実行をスイッチします。このコンテキストのスイッチは素早く周期的に行われ、ユーザーには同時に並行して実行が行われているように見えます。

例えば:

- 最初の CPU サイクル: CPU はスレッド A から最初の実行可能命令を受け取ります。
- 二番目の CPU サイクル: CPU は スレッド A から二番目の実行可能命令を受け取ります。
- 三番目の CPU サイクル: 二番目の命令実行が一時停止され、I/O 処理を待ちます。この時点でオペレーティングシステムは、実行準備ができている次の異なるスレッド B からの命令に実行をスイッチします。
- 四番目の CPU サイクル: CPU はスレッド B から二番目の命令を受け取ります。
- 五番目の CPU サイクル: 一時停止されたスレッド A の命令がデータを受け取り、スレッド A の次の命令が実行されます。

CPU サイクル 1 CPU サイクル 2



最初の CPU サイクルの間、スレッド 1 が実行されます。二番目の CPU サイクルではスレッド 1 の実行が一時停止され、オペレーティングシステムはスレッド 2 に実行を切り替えます。

コオペラティブスレッドの偉大な点のひとつは、バックグランドで長いタスクの実行を行わせている間も、そのアプリケーションがユーザー入力を受け付けることができるということです。例えばオペレーティングシステムがファイルマッピング処理を行っている間も、ユーザーはこれらのファイルを扱うことが可能です。

コオペラティブスレッドの欠点は、停止すべき状態になったときに、制御を渡すかどうかがスレッド自身に依存している点にあります。これはリソースを待っているスレッドが利用可能になった時に問題を発生させることがあります。

プリエンプティブスレッド

プリエンプティブスレッドの実行は共有リソースに影響しません。これによりオペレーティングシステムは同時に複数のスレッドを実行できます。そのようなスレッドの実行は I/O 処理からブロックされません。すべてのプリエンプティブスレッドはその実行に等しくプロセッサータイムを持ちます。スレッド実行の最小時間は 1 CPU サイクルです。シングルコアアーキテクチャーでは CPU サイクル毎にスレッド間を切り替えるコンテキストがあります。

例えば:

- 最初の CPU サイクル: CPU はスレッド A から一番目の実行命令を受け取ります。
- 二番目の CPU サイクル: CPU はスレッド B から一番目の実行命令を受け取ります。
- 三番目 CPU サイクル: CPU はスレッド A から二番目の実行命令を受け取ります。
- 四番目 CPU サイクル: CPU はスレッド B から二番目の実行命令を受け取ります。

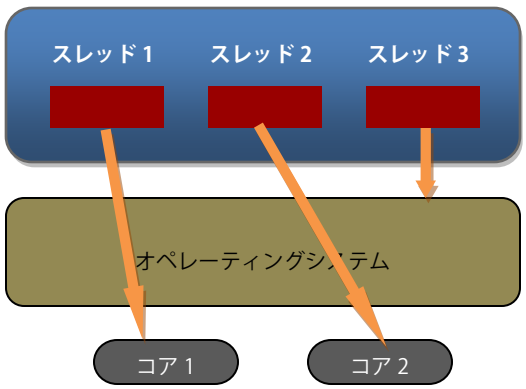
シングルコアアーキテクチャーにおいて、プリエンプティブスレッドはコオペラティブスレッドのように、ユーザーからは並行して実行されているかのように見えます。異なる点は、共有リソースは更新されず、実行はスケジューラーの実行リストの順番に依存しないという点です。

プリエンプティブスレッドでは、オペレーティングシステムが実行をすべてのコアに分割することで、マルチコアシステムを利用できます。これによりパフォーマンスの向上が見込めます。すべての CPU サイクルにおいて、オペレーティングシステムはコア間のコンテキスト切り替えを行います。

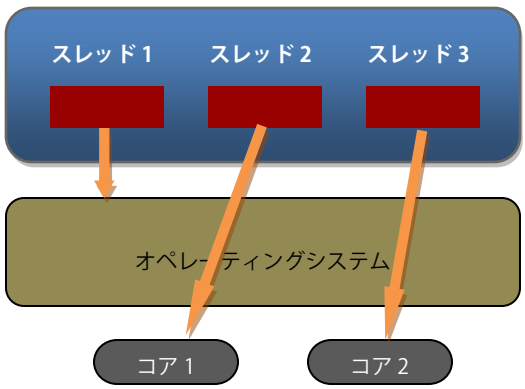
マルチコアアーキテクチャーでは、実行は以下のように行われます:

- 最初の CPU サイクル: スレッド 1 とスレッド 2 は一番目と二番目のコアで実行されます。
- 二番目の CPU サイクル: スレッ 1 が一時停止され、二番目と三番目のスレッドがコア 1 とコア 2 で実行されます。
- 三番目 CPU サイクル: スレッド 2 が一時停止され、スレッド 1 とスレッド 3 がコア 1 とコア 2 で実行されます。

CPU サイクル 1



CPU サイクル 2



プリエンプティブスレッドは、いつコンテキストの切り替えを行うかをオペレーティングシステムが決定できるという点で、最も先進なアプローチと考えられます。欠点は、正しくないときにコンテキストの切り替えをオペレーティングシステムが行ってしまうかもしれず、コオペラティブスレッドでは避けられている負の効果が表れる可能性があるということです。

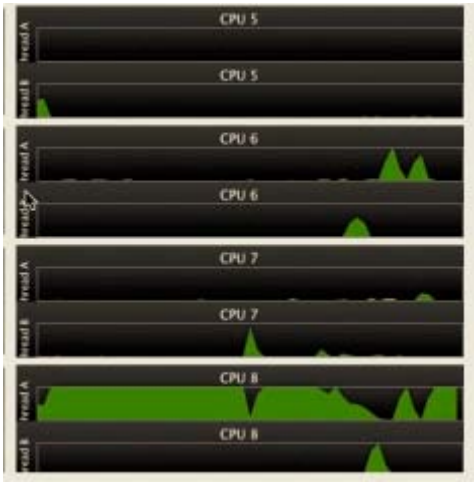
4D v11 SQL における新しいマルチスレッドモデル

4D におけるマルチスレッドモデルはプリエンプティブスレッドに基づいています。この場合、4D はマルチコアマシンの利点を利用することが可能です。プリエンプティブスレッドはオペレーティングシステムから処理され、利用状況に基づきそれぞれのコアに実行を割り当てるのはオペレーティングシステムの役割です。

他方、4D のコオペラティブスレッドは 1 つのコア上で実行され、それらを管理するのは 4D です。

4D スタンドアロンでのマルチスレッド

基本的なルールとして、4D コードを含んデータり、ユーザー入力を受け入れるようなプロセスはコオペラティブスレッドで実行されます。つまりすべてのユーザーインターフェースや 4D ランゲージの処理はコオペラティブスレッドで実行されます。Web サーバーと DB4D エンジンもコオペラティブスレッドで実行されます。このことは以下のテストで簡単に確認できます。4D のスタンドアロンとクライアント/サーバー環境で ORDER BY を実行します。スタンドアロンで、ORDER BY はコオペラティブスレッドで実行されます。クライアント/サーバーではプリエンプティブスレッドで実行されます。以下はスタンドアロンの 4D で ORDER BY を実行したときの CPU アクティビティのスクリーンショットです。



ご覧の通り、実行は 1 コア上で行われています。スタンドアロンの 4D ではデータベースエンジンのすべての呼び出しは 1 つのコオペラティブスレッド上で実行されます。実際スタンドアロンでは、ごく少数のプリエンプティブスレッドが使用され、ほとんどのコードはコオペラティブです。インデックス、データキャッシュ、SQL サーバーへのすべての外部呼出しがプリエンプティブスレッドです。4D はプロセス毎にプリエンプティブスレッドを作成する必要はなく、実行を制御する必要もありません。すべてのプリエンプティブスレッドはオペレーティングシステムに渡され実行されます。

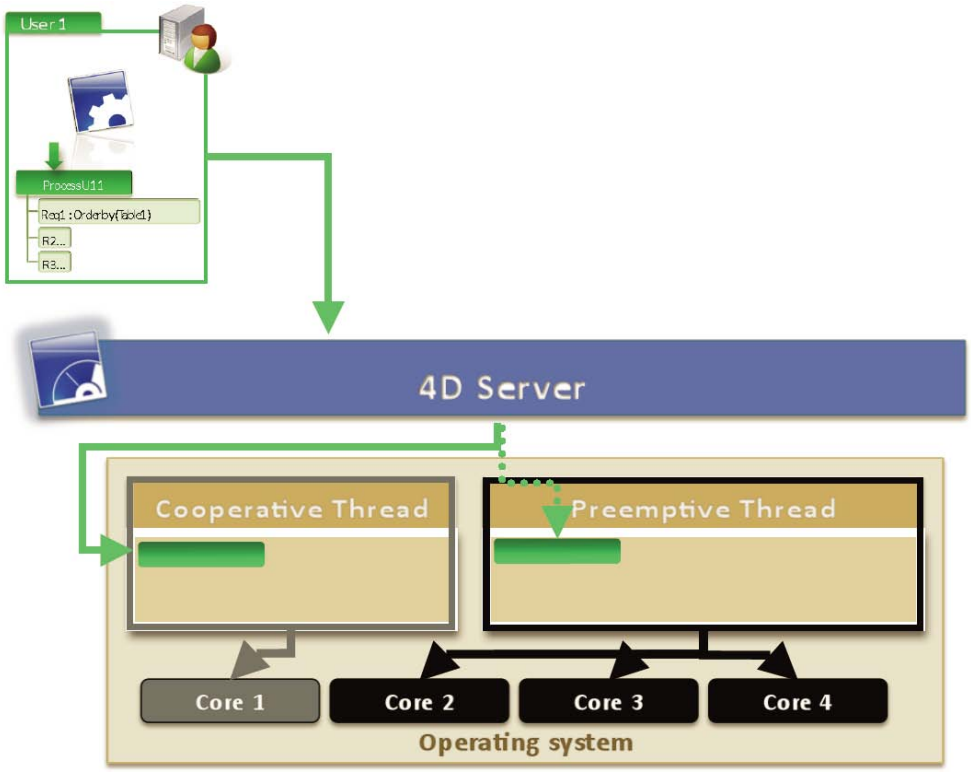
データベースアプリケーション開発中、データベースのデータファイルはさほど大きくなく、バックアップのインデックスなどの時間のかかる処理も行いません。よって開発フェーズでは少ないコアのマシンでも十分です。

新しいマルチスレッドモードを完全に利用できるのは 4D Server v11 SQL です。次の節では 4D Server おけるマルチスレッドについて説明します。

4D Server でのマルチスレッド

新しい非ローカルプロセスがクライアント上で開始されるたびに、対応するプロセスがサーバー上でも実行されます。このプロセスはコオペラティブスレッド上で実行され、4D が 2 つのプロセス間の同期を管理します。これに加え、4D Server はプリエンプティブな双子のプロセスをすべてのクライアントプロセスごとに作成します。

4D Server v11 SQL と 4D クライアント間で通信がどのように行われるのか見てみましょう。以下の 2 つの図はプロセスを表しています。最初は 1 つのクライアントがサーバーに接続しています。新しいプロセス"U11"がクライアントマシン上で開始され、2 つの対応するプロセス (コオペラティブスレッド (線) とプリエンプティブスレッド (点線)) がサーバー上で作成されます。



コオペラティブスレッドは Delay process や Get process variable などのアプリケーションリクエストを処理します。

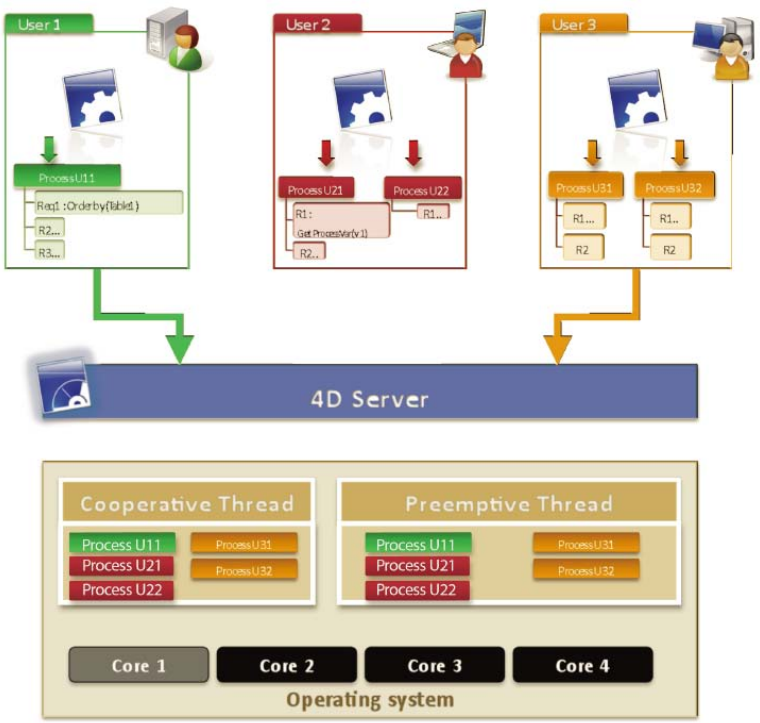
プリエンプティブスレッドは並び替えやクエリ、インデックス処理などすべてのデータベースエンジンリクエストを処理します。実行は複数のコアに分割されます。それに加え、Begin SQL/End SQL による SQL リクエストを処理する追加のプリエンプティブスレッドが作成されることもあります。

例えば:

```
C_TEXT($stmnt)
ARRAY TEXT(firstName_a;0)
ARRAY TEXT(lastName_a;0)
$stmnt:=”SELECT FirstName, LastName FROM Reople INTO :firstName_a,:lastName_a)”
Begin SQL
    EXECUTE IMMEDIATE($STMNT)
End SQL
```

新しいプリエンプティブスレッドは Begin SQL/End SQL の呼び出しで作成されます。すべての実行可能命令はスレッドの実行可能リストに追加されます。

次の図は二人目、三人目のクライアントが 4D Server に接続した状態を示しています。



ポート 19813 を通じて送信されるすべてのクライアントリクエストは 1 つのコオペラティブスレッドで実行されます。さらにすべての Web サーバーへのリクエストも同じコオペラティブスレッドで実行されます。すべてのコオペラティブ命令は 4D スケジューラーにより処理されます。

Note: 4D スケジューラーに関する詳細はテクニカルノート"The 4D v11 SQL Scheduler"を参照してください。http://kb.4d.com/search/assetid=75963

すべてのプリエンプティブスレッドは 4D Server のポート番号 19814 で受け付けます。4D が 1 つのスレッドを作成するのではなく、オペレーティングシステムに渡され、異なるコアで実行されます。すべてのプリエンプティブプロセス毎に、4D は 1MB のメモリースタックを割り当てます。このスタックサイズは SET DATABASE PARAMETER のセレクター53 で変更できます。このサイズを増やし、同時に多数のクライアントが接続すると、プリエンプティブスレッドに割り当てられるメモリー量が増大し、アプリケーションパフォーマンスが悪化することがあります。デフォルトサイズは 1MB で、512KB や 256KB に減らすことができます。

以下のコードはスタックサイズを 1MB から 256KB に設定します:

```
SET DATABASE PARAMETER(53; 262144)
```

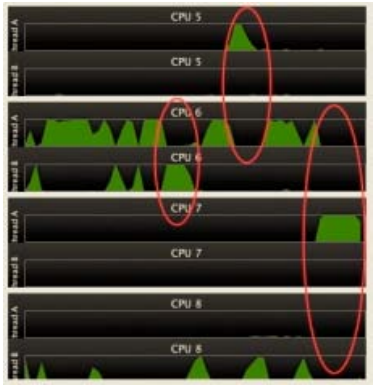
マルチスレッドモデルのサンプルを見てみましょう。

最初のテストの目的はマルチコアマシン上でコオペラティブスレッドがどのように実行されるかをみることです。

以下のコードはコオペラティブスレッドで実行されます。

```
ARRAY TEXT($testArray;20000)
C_LONGINT($i) `ループインデックス
For($i;1;Size of array($testArray))
   $testArray{i}:=String(Random;"000000000000000000000000")
End for
Repeat
   SORT ARRAY ($testArray)
   IDLE
Until (Macintosh option down)
```

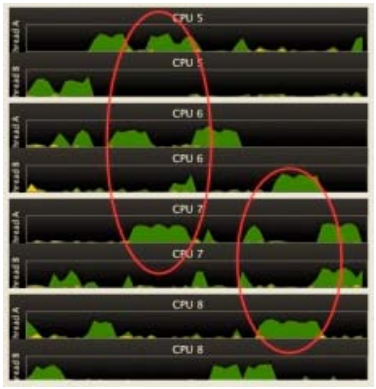
この 4D コードは"サーバー上で実行"メソッドプロパティをチェックすることで、サーバー上で実行されます。以下のスクリーンショットはマルチコアのマシン上で、このコードがどのように実行されたかを示しています。命令はコオペラティブスレッドで実行されています。



赤丸はオペレーティングシステムがコア間でどのようにコンテキストの切り替えを行ったかを示しています。スレッドの実行は異なるコアに移動されていますが、一度には 1 つのコアしか使用されません。1 つのプロセスが一時停止されると、次に実行可能なプロセスが異なるコアで実行されます。

次は 4D でレコードの並び替えを行うプリエンプティブスレッドの例題です。サーバー側ではレコードの並び替えはプリエンプティブスレッドで行われます。サーバーデータベースエンジンの呼び出しはすべてプリエンプティブスレッドで行われます。これはスタンドアロンの 4D と異なる点です。以前のバージョンの 4D ではすべてのサーバープロセスがコオペラティブスレッドで実行されていました。

以下のスクリーンショットはマルチコアマシン上でマルチスレッドがどのように実行されているかを示しています。赤丸は 1 つのプリエンプティブスレッドがマルチコア上でどのように実行されたかを示しています。



実行は複数のコアに振り分けられています。どのように分割を行うかはオペレーティングシステムが決定します。同じスレッドが異なるコア上で実行されます。

並び替えが Execute on server コマンドでストアドプロシージャーから呼び出されると、マルチコアの恩恵を受けることはできません。ストアドプロシージャーはコオペラティブスレッドで実行されるからです。以下のスクリーンショットを見てください。

実行は 1 つのコア上でのみ行われています。

ストアドプロシージャーとサーバー上で実行メソッドプロパティ

先に示した通り、Execute on server コマンドで作成されたストアドプロシージャーは 1 つのコオペラティブスレッドで実行されます。サーバー上に独自の変数テーブル、カレントレコード等を持つ新しいプロセスが開始されます。SET PROCESS VARIABLE や VARIABLE TO VARIABLE コマンドを使用して異なるプロセスから変数にアクセスすることができます。言い換えると、ストアドプロシージャーで実行されるコードはスレッドセーフではありません。他のスレッドからプロセス変数を変更させないようにするために、すべてのストアドプロシージャーは 1 つのコオペラティブスレッドで実行されます。複数コアの恩恵を受けることはできません。

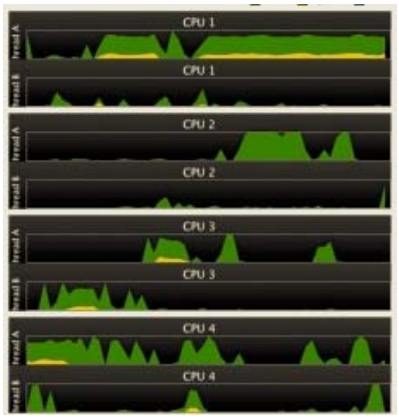
サーバー上でコードを実行するための方法として、4D v11 SQL では新しいメソッドプロパティ"サーバー上で実行"を使用できるようになりました。

Note:　このプロパティはプロジェクトメソッドでのみ利用できます。

サーバー上で実行プロパティはサーバー上の双子プロセスを使用します。メソッドコードはストアドプロシージャーと同様に実行されますが、サーバーに最適化されたコマンドはプリエンプティブスレッド

で実行され、マルチコアの恩恵を受けることができます。以下はサーバー上で実行を利用した場合の例題です:

SELECTION TO ARRAY を呼び出してみましょう。最初は"サーバー上で実行"プロパティを有効にし、次に無効にします。



上のスクリーンショットで見られるとおり、実行は複数コアに振り分けられています。

"サーバー上で実行"を無効にした場合、SELECTION TO ARRAY はコオペラティブスレッドで実行されます。

コードを含むメソッドの"サーバー上で実行"プロパティが有効にされている場合でも、トランザクション、レコードロック、プロセスセット、プロセス命名セレクションは 1 つのコオペラティブスレッドで実行されます。

これらはクライアントとサーバー間で共有されるリソースです。異なるクライアントはそれらを個別に操作することができます。言い換えればこれらはスレッドセーフではなく、プリエンプティブスレッドで実行することはできません。そこで、これらは常に 1 つのコオペラティブスレッドで実行されます。

トリガー

トリガーはサーバー上で実行されます。トリガーは呼び出されたクライアントと同じコンテキストで実行され、"サーバー上で実行"メソッドプロパティと同様、新しいプロセスは作成しません。異なる点は、トリガーは (レコード作成、更新、削除時) 特定のイベント時に実行されるということです。またトリガの実行は他のトリガーによって遅延されることがあります。このためトリガーは常にコオペラティブスレッドで実行されます。通常プリエンプティブスレッドで実行されるコマンドに関連するすべての実行命令はコオペラティブスレッドで実行されます。トリガーはマルチコアの恩恵を受けることはできません。マルチコアシステムにおいては、可能な限りトリガーの利用は最小限に抑えたほうがよいでしょう。すべてのデータクセスコマンドはトリガーコードから取り除くべきです。

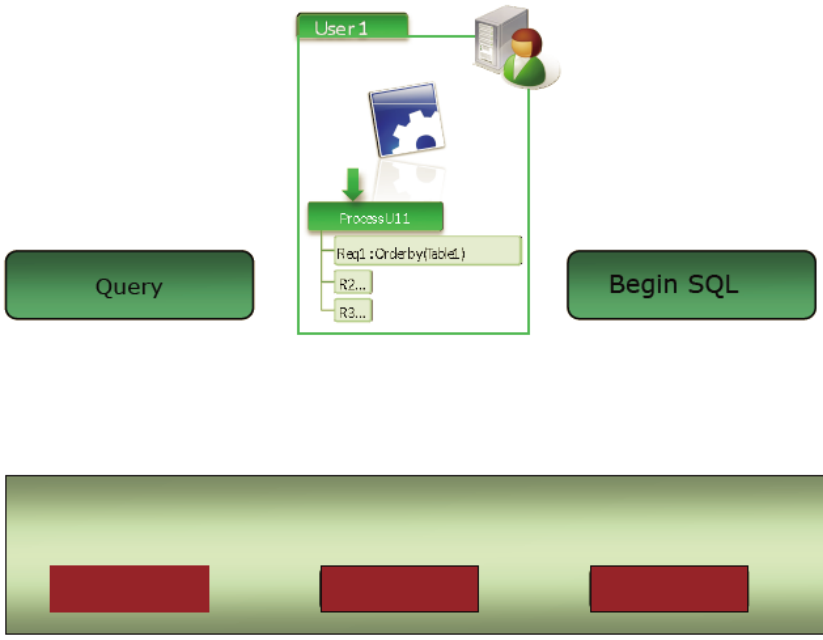
インデックス、データキャッシュ、バックアップ処理

インデックス、データキャッシュ (バッファーのフラッシュ) そしてバックアップ処理はデータベースアクセス命令であり、複数のプリエンプティブスレッドで動作します。実行結果はスレッド間で共有されるリソースに影響せず、異なるプリエンプティブスレッドで実行しても安全です。マルチコアアーキテクチャーにおいて、以前の 4D Server バージョンに比べその実行はより速くなりました。新しいプリエンプティブスレッドにより、ユーザーは作業を続行することができ、同時にデータベース処理をバックグランドで実行することができるようになりました。例えばバックアップ処理中にユーザーの I/O がブロックされることはなくなりました。

4D SQL サーバー

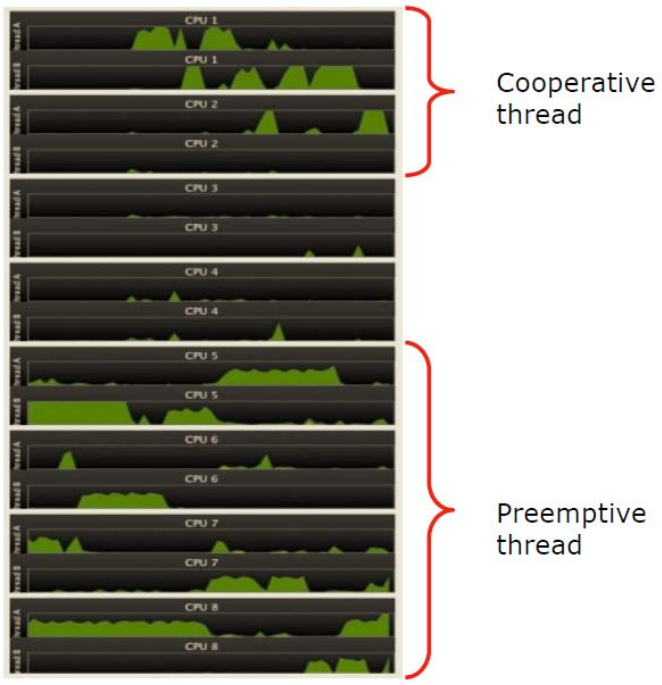
4D SQL サーバーのプロセスは 4D スタンドアロンおよび 4D Server でプリエンプティブスレッドで実行されます。ポート番号 19812 で受信したすべてのリクエストは複数のコアに実行が振り分けられます。また Begin SQL/End SQL が呼び出されると新しくプリエンプティブスレッドが作成されます。

例えば:



検索命令はサーバーのポート番号 19814 に送信され、プリエンプティブスレッドで実行されます。Begin SQL/End SQL 命令はポート番号 19812 に送信され、2 つのプリエンプティブプロセスで実行されます。両方とも DB4D リクエストですが、異なる API を通しています。

以下のスクリーンショットはコオペラティブスレッドとプリエンプティブスレッドを同時に実行した様子を示しています。



コオペラティブスレッドは一番目と二番目のコアで実行され、プリエンプティブスレッドは五～八番目のコアで実行されています。

Note: *コオペラティブスレッドとプリエンプティブスレッドは同時に同じコアで実行されることはありません。それらは常に異なるコアで実行されます。*

この例は 4D のマルチスレッドを理解することがとても重要であることを示すものです。例えば、データベースの操作はプリエンプティブスレッド内で実行されるデータベースエンジン命令に分解され、結果がコオペラティブスレッドに適用されることがあります。プリエンプティブスレッドの実行はマルチコアのおかげでとても素早く終了します。

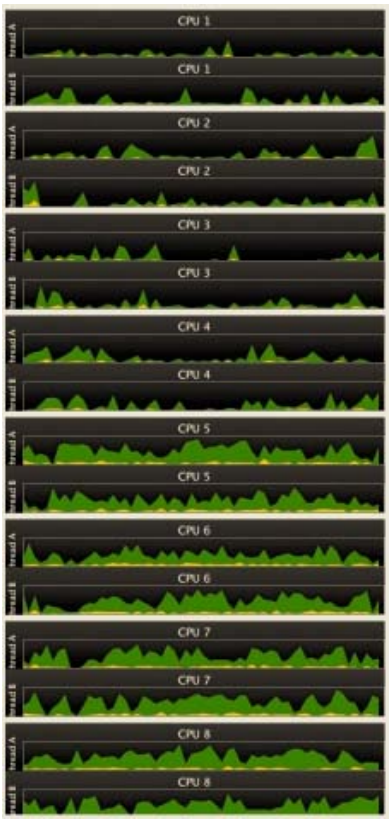
以下はコオペラティブとプリエンプティブスレッドが 4D スタンドアロンと 4D サーバー/クライアント環境それぞれでどのように使用されるかを示す概要図です。

	4D Local		4D Remote/Server	
Thread	**Cooperative**	**Preemptive**	**Cooperative**	**Preemptive**
User interface processes	✓		✓	
4D Language processes	✓		✓	
Stored procedures			✓	
HTTP Server	✓		✓	
Application Server			✓	
Index Builder		✓		✓
Flushing Data		✓		✓
SQL Server		✓		✓
DB4D	✓			✓

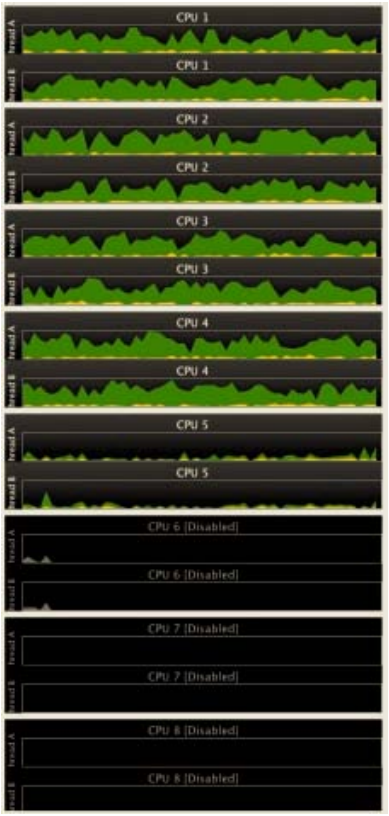
拡張性

4D Server v11 SQL では大きな拡張可能性を備えていて、4D Server 自身が増大する作業量に対応することができるということのほか、サーバーに接続するユーザー数の増加にも対応できることを意味します。同時に複数のデータファイル処理を複数のクライアントが開始できるため、マルチコアの恩恵を完全に受けることができます。
あるクライアントプロセスが他のプロセスに優先されることはありません。言い換えればクライアントのプロセスは同時に実行され、オペレーティングシステムがその実行を担当します。

次の例では 4D Server v11 SQL をマルチコアアーキテクチャーで実行することが重要であることを示すものです。3 台のクライアントが接続されたサーバーで並び替えを行ってみましょう。



オペレーティングシステムはスレッドの実行を複数のコアに分散させています。CPU の稼働率が変化するのを見るために、3 つのコアを無効にしてみましょう。



他のコアの稼働率が上奏したことが分かります。これはアプリケーション全体のパフォーマンスを低下させます。言い換えればサーバーに接続されるクライアントが多くなるほど、また同時に実行するデータベース命令が多くなればなるほど、多くのコアを実装した環境がパフォーマンス向上のために役に立ちます。

まとめ

4D v11 SQL と 4D Server v11 SQL はマルチスレッドアプリケーションであり、マルチコアアーキテクチャーの恩恵を受けることができます。4D の開発者はデータベースの速度向上のために、4D やオペレーティングシステムがどのように異なるスレッドの実行を処理するのか、またどのような調整が必要なのかについて知っておく必要があります。また CPU サイクルを見ることで、オペレーティングシステムがどのようにこれらのタスクを実行しているのか、またコードが本当に新しいマルチスレッドモデルを使用しているのかについてのフィードバックが得られます。

利点は:

- すべてのデータクセス、インデックス、およびデータベース処理や SQL エンジンの呼び出しは、即座にマルチスレッドの恩恵を得ることができます。
- マルチタスク – 異なるプロセスを同時に実行できます。あるプロセスはユーザー入力を待ち、同時に他のスレッドは空きリソースを利用して実行を続けることができます。これにより全体のパフォーマンスが向上します。バックアップやインデックス処理のために現在行っていることを停止させる必要がなくなるため、このことはユーザーにとって重要です。
- 拡張性– 複数のクライアントがサーバーに接続していたり、複数のデータベース処理を同時に行うといった場合、これらの処理を複数のプリエンプティブスレッドで実行させることができます。CPU 利用率はより並行的になり、CPU 数により変わってきます。