Technical Note 04-39
プロセス間で通信する

By Jean-Yves Fock-Hoon QA Manager, 4D, Inc.
Technical Note 04-39

（原題: Exchanging Data Between Processes）

概要

データベースで複数のプロセスが起動しており、プロセス間でデータをやり取りして互いのプロセス変数に値を代入する必要が生じることがあります。これは一見、簡単なようでなかなか厄介な作業です。この Tech Note では、プロセス間の通信をなるべく簡単にとどめるためのノウハウを取り上げています。

導入

今回、使用するのは次の 3 種類のコマンドです。

SET PROCESS VARIABLE-特定のプロセスにおけるプロセス変数に値を代入する
GET PROCESS VARIABLE-特定のプロセスにおけるプロセス変数の値を取得する
VARIABLE TO VARIABLE-特定のプロセスにおけるプロセス変数に値を代入する

コマンドについてはランゲージリファレンスに詳述されています。注意しなくてはいけないのは、変数は代入元、代入先の両プロセスで定義されていなければならないという点です。SET PROCESS VARIABLE と VARIABLE TO VARIABLE を比較した場合、後者だけが配列を扱えるという点が異なっています。

上記のコマンドは、変数に値が代入されるまでプロセスを待機させる場合に有効です。たとえば、複数のプロセスを同時に実行しているアプリケーションで 4D を終了したいとします。データ保存が未完のプロセスがあるかもしれないので、すべてのプロセスを強制終了するわけにはいきません。むしる、そのような場合はセマフォを使用し、各プロセスはセマフォがあがったことを確認してから自らを終了するようにします。同様の方法としては、<>Quit4D のようなインタープロセス変数を用意し、各プロセスが終了の判断に使用するというものが挙げられます。

これらの方法はすべてのプロセスを終了する場合にはよくても、特定のプロセスだけを終了するには向いていません。プロセスごとに担当のインタープロセス変数を用意する方法もありますが、プロセスの数が予測できない場合には対応できません。プロセス番号のインタープロセス配列を管理するという方法も現実的ではありません。プロセスが生まれた

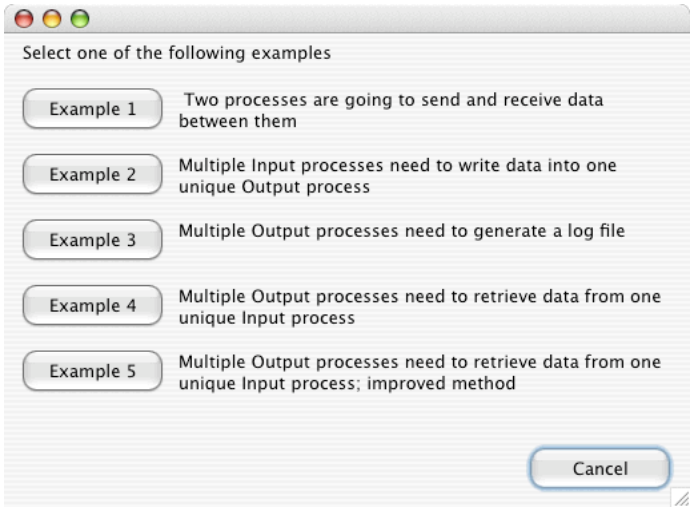
り消えたりすれば、プロセス番号が再利用されるからです。

むしろ、各プロセスが自らのプロセスを確認するという仕組みのほうがシンプルであるといえます。あるプロセスが他のプロセスを終了したいと思えば、相手プロセスのプロセス変数に値を代入するだけで済むからです。

インタープロセス変数は非常に便利ですが、すべてのプロセスに共通であるがゆえ、特定のプロセスだけに渡したい情報を運ぶには不向きです。場合によっては、相手のプロセス編数を直接操作したほうが実際的です。

このような訳で、冒頭に挙げた 3 種類のコマンドが必要になってきます。プロセスを終了させるという上記の例では、いわば打ちっ放しのように値を代入するだけで済みました。では相互にデータをやり取りしなければならない場合はどうでしょうか。そのような場合肝心なのは、何といっても通信の同期をとるということです。

例題 1:単純な通信



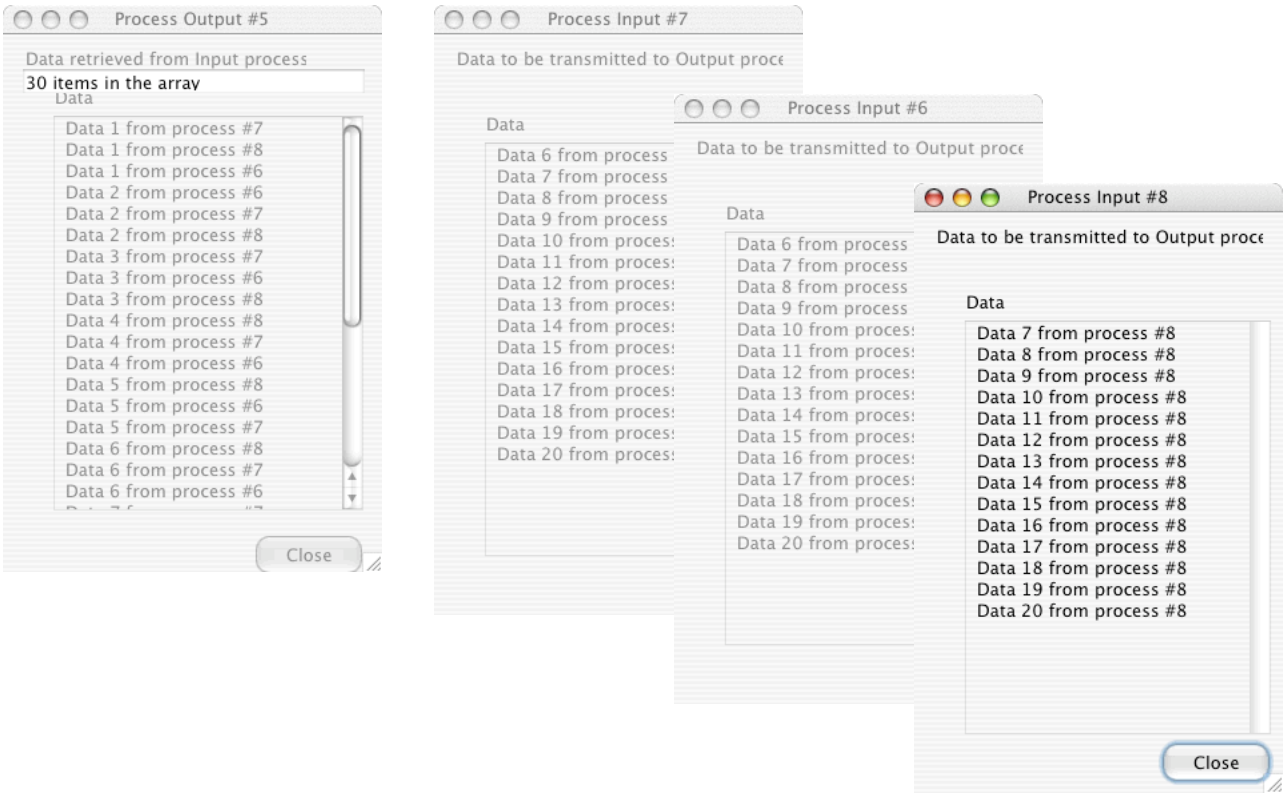
Example1 ボタンをクリックすると、プロセスがふたつ起動します。Process One と Process Two は相互に会話し、その様子がウィンドウに表示されます。

それぞれを動かしているメソッドは M_ProcessOne および M_ProcessTwo です。Process One は Process Two が初期化をしている間、<>Ready を監視して待っています。相手の準備ができ次第 Process One は次の動作へ進みます。このように相手の用意ができてから先へ進むというのがこの例題のポイントです。

Process One は、SET PROCESS VARIABLE コマンドでメッセージを vReceivedMessage に代入します。Process Two のものであるこのプロセス変数は、当初、空の文字列です。Process Two は空の文字列以外の値が確認された時点で Process One に対してメッセージを送ります。このように双方が vReceivedMessage を媒体として交替でメッセージを送りあっているので、この通信は同期がとれているといえます。

例題 2: 少し高度な通信

今度は、相手からの応答がなくても通信ができるという例です。Example2 ボタンをクリックすると、データを発信するプロセスが同時に 3 個起動します。これらは共通の相手プロセスと通信しており、誰がデータを発信しても Process Output が受け取るという仕組みです。



Process Output が Example1 のように各プロセスに対して待機すれば、最初のプロセスから入力があるまでは次のプロセスに順番が回らないことになります。そこで今度はインタープロセス配列をバッファとして使用します。各プロセスは直接プロセスと通信する代わりに、バッファに対してデータを書き込みます。書き込みの競合を避けるためにはセマフォを使用します。この場合、配列の要素として書き込みがスタックされることになります。

サンプルでは、要点を明確にするために DELAY PROCESS で Process Output の動作を遅

くしています。このため Process Output が高速でそれぞれのプロセスを確認するというプログラミングは不可能です。もうひとつの工夫は Process Output がバッファを取り込む間はセマフォ（入力プロセスの競合を避けるために使用するのと同じもの）をあげ、取り込み終えたら戻しているという点です。Process Output がデータを処理している間、各プロセスはバッファに書き込むことができます。

さらに制限をかけるために、それぞれのプロセスでダイアログウィンドウを開き、ウィンドウに経過を表示させることにしました。4D はアイドル時にウィンドウを再描画するので、再描画には意識的なコーディングが必要です。データを発信するプロセスの場合 On Timer イベントでセマフォをあげ、データをバッファに書き込み、セマフォを解放して次の On Timer まで待機するようにします。同様に Process Output も用事がない間はアイドルにすることで再描画をすることができます。データを発信するプロセスがバッファに書き込んだ直後に CALL PROCESS で Process Output を起こすのが合理的ですが、問題はこのプロセスが遅くされているという点です。4D は、多数のフォームイベントをためこむことができないので On Load、On Clicked あるいは On Outside call の最中に発生した On Timer は考慮されません。とはいえ On Outside Call ばかりを気にしていては本業であるバッファの処理ができないので SET TIMER(30)を設定しています。この結果、ウィンドウの再描画とバッファの処理をバッファが空になるまで繰り返すことができます。

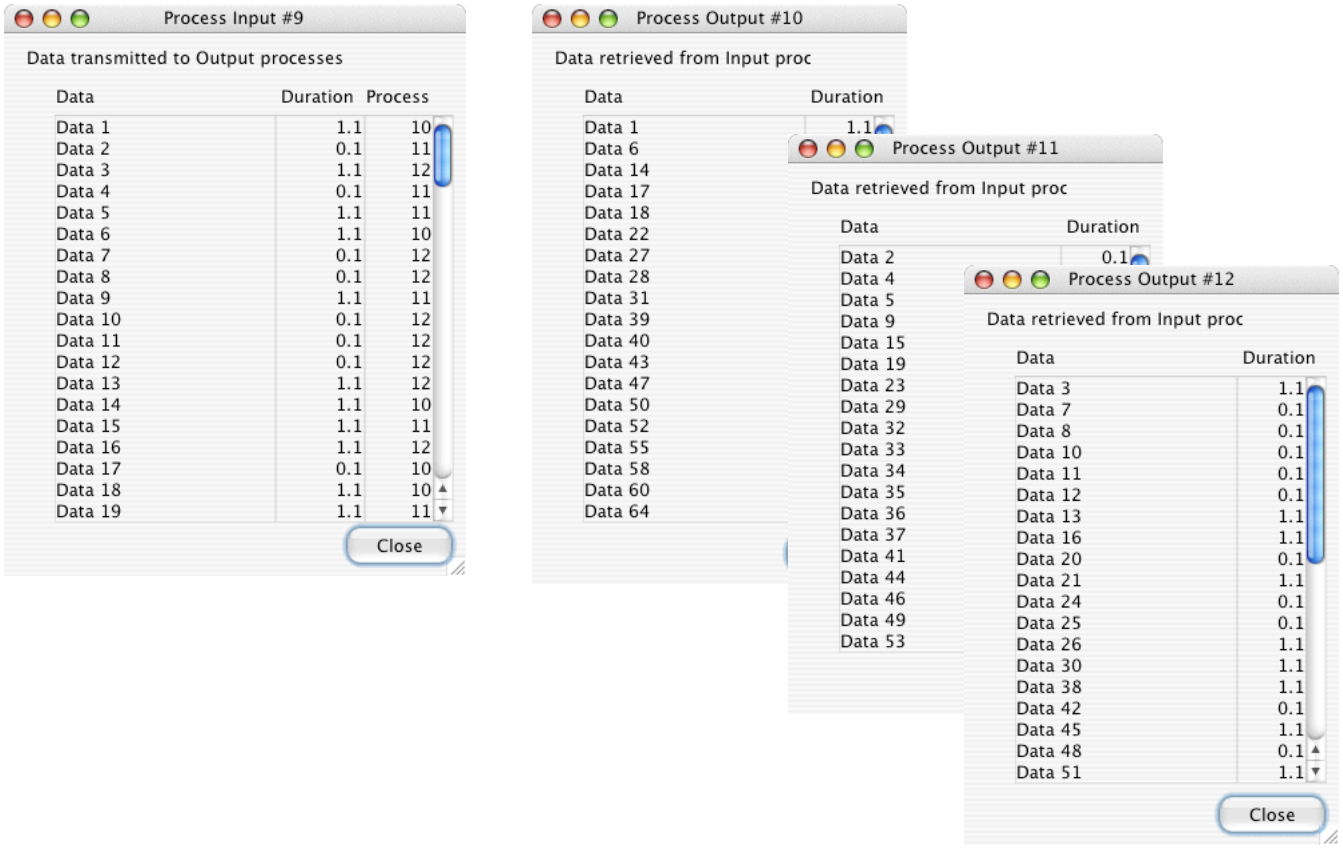
プロセスが CPU 時間を浪費するようなことは避けるべきです。ウィンドウにプロセスを反映させるにはアイドル時間が必要なので On Timer や On Outside call を利用します。プロセス同士の通信はいつでも同期をとれるというわけではありません。非同期の通信にはバッファ（この場合はインタープロセス配列）のような仕組みを利用します。プロセスの同期がとれていないことはよくあるエラーのひとつです。たとえば、ナビゲーションボタンをフローティングウィンドウで提供し、レコードを別プロセスのウィンドウで表示するようなインタフェースで、次レコードボタンをクリックすると CALL PROCESS でレコード表示ウィンドウが反応するような場合、レコード表示が再描画されるよりも素早くクリックを続けるとプロセス同士の認識が食い違うような事態に進展しかねません。

例題 3: ログファイルシステム

同じ通信でもウィンドウを使用しなければ CALL PROCESS は不要です。複数のプロセスがセマフォを利用してバッファに書き込み、共通の出力プロセスに対してデータを送るという仕組みは、ログの記録に最適です。出力プロセスはアプリケーションの各プロセスから受け取ったデータをログとしてテキストファイルに書き出します。ログを記録することによって、同期の問題を検出したり、頻繁に実行されるメソッドを調べたり、クラッシュの経過を解析するさえできるかもしれません。いずれにしても出力プロセスの動作はできるだけ軽くするように努めます。

例題 4：逆の構図

今度は、単一のプロセスから発信されたデータを複数のプロセスが受け取って処理するという例です。データは番号と時間で構成されており、時間はデータの処理に要する時間を想定しています。同期にはセマフォと配列、加えてプロセス変数コマンドを利用します。



複数の Process Output が同時にアクセスをしないようにセマフォを確認させています。それぞれの Process Output はデータを受け取る前に自らのプロセス番号を告げ、配列から一番手前のデータを削除し、次のデータをプロセス変数に取り込んでいます。Process Input は GET PROCESS VARIABLE でこの相手プロセスを調べて DELAY PROCESS を実行し、次の Process Output に順番が回るようにしています。

これはセマフォ、および 2 プロセス間セマフォとしての変数の簡単な使用例です。はじめに GetNextItem が False の状態で Process Output が動いています。Process Output はデータを処理し、一時停止前に GetNextItem を True にします。この変数を調べることによって処理が完了したことが分かるので、Process Input は結果を受け取り、セマフォを解放して他の Process Output がデータを受け取れるようにしています。

例題 5：例題 4 を改良

それぞれの Process Output がセマフォを使用して可能なときに配列の要素を GET PROCESS VARIABLE で取得するようにすれば Process Input は全体の処理が完了までスリープしていればよいので効率的です。