生産ラインにおける制御用プログラム開発の効率化

Improving Productivity of Software Development for Assembly Line Control

山田 周司 [1)]
Syuji YAMADA

1) 有限会社 エスワイシステム (〒939-8045 富山市本郷町 281 番地の 2
E-mail:sy_yamada@nifty.com)

ABSTRACT. Generally, when source code for a production control system is described, handlers of each event such as message notification tend to be implemented in fragments. And use of more than one process or thread usually necessitates the division of source code into multiple source files. To get an idea of the actual behavior of such split-up source code, we usually have to do laborious work, bringing message sources and message destinations into correspondence, linking fragments, and thereby identifying the global flow of processing. This work process has been causing significant productivity degradation in system development, modification, and maintenance. To solve the problem, I suggest a new approach using a 'packet.' In this approach, instead of simply exchanging data among threads, information about a sequence of operation steps (which would conventionally be performed in multiple threads) as well as the required data are placed in individual packets, and then packets are executed. Under this project, I designed a mechanism for generating packets from script and executing the operation steps contained in packets. Using this mechanism, I designed a tool for developing packet-based applications.

1. 背景

工場の生産ラインの制御, データ収集･監視装置, 検査装置にはＰＬＣ(シーケンサ)やパソコンが使用されているが機械に近い処理ほどＰＣＬの採用が多くなる.

Ｗｉｎｄｏｗｓパソコンではマルチスレッド, マルチプロセスをサポートするＡＰＩが存在するので, リアルタイムでの応答速度が問題とならない用途に対しては, 制御用のアプリケーションソフト開発が可能である. しかしパソコンで制御用プログラムを開発するには十分な設計とテストが必要であり多大な開発工数がかかる.

高価なシステムや大手メーカーのラインにはパソコン(機器組み込み型も含む)が採用されている事例をいくつも見つけることができるが, コスト面の制約が厳しい装置にはパソコンがほとんど採用されていないといったことから, パソコンでのソフト開発コスト低減が大きな課題といえる.

以上からパソコン上で制御用プログラムを簡単に開発できるツールへの潜在的なニーズがあるものと見られる.

2. 目的

本プロジェクトではマルチスレッド･マルチプロセス対応の制御用プログラムの開発をパソコン上で効率よく, すなわち低コストで開発できるツールを作成する.

3. システム開発の新しいモデル

プロセスとスレッドの違いについて簡単に説明すると, 1つのアプリケーションプログラムに対応するものがプロセスであり, 1つのプロセス中には複数のスレッドを同時に実行させることができる. 以下ではプロセス間, スレッド間をまとめ,「スレッド間」と表現する.

従来のスタイルでスレッド間のメッセージ送受信を行なう場合, メッセージ通知で送信する領域にはデータのみが格納され, これを受信し領域からデータを取り出した後, または取り出したデータによる処理を行っ

た後すぐに領域は廃棄される．すなわちメッセージの送受信はスレッド間のデータ伝達とタイミングを通知するだけの手段である．

今回提案するスタイルは次のようなものである．

- 領域（パケット）を単にデータの格納エリアとして使用するのではなく，スレッド間に渡る一連の処理に必要な手順の指示（スクリプト）と関連するデータの双方を格納する．
- データ格納エリアは1回のデータ受け渡しに必要なエリアではなく，一連の処理が完了するまでに必要なエリアをあらかじめ確保するものとする．
- スクリプトを処理する機構は各スレッドが持つものとし共通の処理ルーチンを使用する．
- 一連の処理を完了するまでパケットはいずれか1つのスレッドに保持される．

以上の改善により，生産ラインのように複数のユニットが協調して作業しワークを順次加工していくといった現場でよく見られるシステムを，プログラム上で簡潔に記述できるようになる．

具体的には以下のような考え方にしたがってモデル化する．

- 1つの機能ユニットについて1つのスレッドを対応させる．このスレッドはパケットを処理できる共通の属性を持つオブジェクトとする．
- 各機能ユニット固有の動作は派生クラスの機能としてそれぞれ記述する．
- ワーク投入（デジタル入力の変化）や，画面のボタンクリック等，一連の処理のきっかけとなる事象が発生したらこれを処理するパケットを生成する．
- 機能ユニットを表すスレッドは，パケットを参照して現在のステップの処理を実行した後，次に実行すべきステップを示すステータス変数をインクリメントする．次に実行すべき命令に他のユニット（スレッド）の処理が出てきたらパケットを次のスレッドにポスト（送信）する．この動作はスレッドの基本的な属性として自動的に処理されるものとする．
- ある機能ユニットが自身の処理を進行する際に他のユニットへ作業を依頼し，完了を待つ場合は新規のサブルーチン的なパケットを生成する．
- 一連の作業が全て完了したらパケットが廃棄される．

以上のような動作の組み合わせとしてシステム全体を記述することとする．この処理方式は伝票を発行して複数の部門間にまたがる処理を進行していくといった企業内の業務に似ている．

4. 開発ツールの構成

本開発環境はスクリプト処理プログラム，パケット処理プログラム，パケット管理ＤＬＬ，デバッグツールから成る．この他に，別途市販のＣ＋＋コンパイラを使用する．

4.1. スクリプト処理プログラム

ユーザが記述したパケット処理のスクリプトを読み込み命令コードを生成する．またスクリプトから呼び出され，スレッドで実行される関数のテンプレートをソースプログラムの中に生成する．対象となるソースプログラムが無い場合には関数のテンプレートを含むソースプログラムが新規に自動生成される．

ユーザは生成された空の関数の中に各スレッドで実行したい処理を記述する．

このスレッドのソースプログラムと以下に述べるパケット処理プログラム，パケット管理ＤＬＬをコンパイルリンクして実行可能なｅｘｅファイルを作成する．（図1参照）

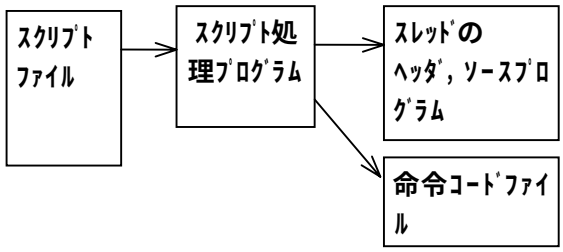

図　1　スクリプト処理プログラム

4.2. パケット処理プログラム

パケットの命令実行，他スレッドへのパケットのポスト，パケットの待ち状態の管理を行なう．（図2参照）

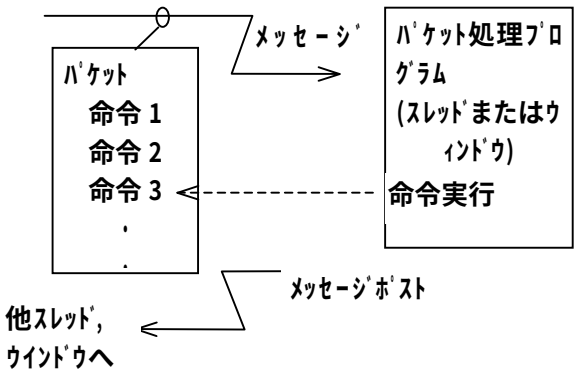

図　2　パケット処理プログラム

4.3. パケット管理ＤＬＬ

パケットの生成･廃棄やエリアの妥当性チェック等の管理をアプリケーション全体に渡って行なう．（図3参照）

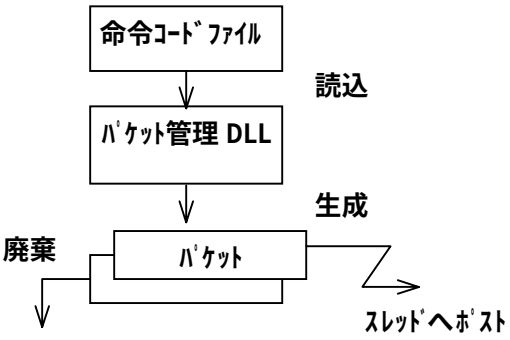

図 3 パケット管理DLL

DLLはシステム起動時に命令コードを読み込み記憶する. パケット生成の要求を受けると, 命令コードをコピーする等してパケットの初期化を行った後に, 最初の命令実行スレッドへポストする.

4.4. デバッグツール

以下の機能を持つ

(1) スクリプトのステップ実行, ブレークポイントの設定機能

本機能はスクリプト単体のデバッグに使用する. スクリプトの中で関数呼び出し以外の箇所は他のパケットと独立に実行されるので, スクリプト単体でのデバッグが効率的である.

(2) デバッグメッセージ表示機能

スレッドのソースプログラム中に記述したTrace文のメッセージ内容を表示する.

パケット相互間の時系列的な関係の確認にはパケットから呼び出される関数の中にTrace文を埋め込む方法が有効である.

(3) DLLの管理情報表示機能

共有メモリの割り当て状況, システム中に存在するパケットの情報, パケットを実行するウィンドウまたはスレッドの情報を表示する.

5. ツールによる開発の流れ

(1) スレッド, パケットの定義

ユーザは, 使用したいスレッド, パケットの名称, IDをプロジェクト管理ファイルと呼ばれるファイルに定義する. 名称はスクリプトの中で使用される. IDはシステムの内部で管理される整数の値である.

(2) スクリプト記述

各パケットのスクリプトをファイル<パケット名>.pktに記述する.

(3) スクリプト処理プログラムの実行

スクリプト処理プログラムはプロジェクト管理ファイル, スクリプトファイルを読み込みスレッドのヘッダ, スレッドのソースファイルを生成する. 生成するファイルにはパケットから呼び出されて各スレッドで実行される関数のテンプレートを生成する.

生成するファイルと同じ名前のファイルが既に存在する場合はそのファイルの中に新しい関数のテンプレートを追加する.

(4) スレッドのソースに生成される関数のテンプレートに記述を追加

外部機器とのインターフェイス処理や, バックグラウンドで実行したい処理等を記述する.

(5) スクリプトの修正

スクリプトを修正した場合はスクリプト処理プログラムを再度実行する.

(6) コンパイルとリンク

コンパイル, リンクの対象は上記(4)により編集した各スレッドのソース, その他のアプリケーション構築に必要なソース, パケット処理プログラムライブラリ, パケット管理DLLのライブラリである.

6. スクリプト

スクリプトはC言語ライクな仕様とする. 処理速度の観点から詳細はスレッドで実行する関数の中で記述することとし, スクリプトで記述する部分は処理の流れが分かり易く記述できればよいという考え方により, 最小限の機能を持たせている.

(1) 変数

スクリプトで変数を宣言するとパケットに領域が確保される.

- 変数の型は bool, long, double の3種類のみとする.
- 変数宣言

  var <型> <変数名> ;

  var struct <型> <変数名> ;

  型として bool, long, double を指定するとスクリプト上で参照, 更新が可能となる.

  その他の char 型, 配列(bool, long, double の配列を含む), 定義済みの型, 及び構造体を指定することも可能とするが, データの受け渡し用エリアがパケットに確保されるのみで, スクリプトからこれを参照することはできない.

(2) 式

演算子と"(" , ")" , 変数, 数値定数の組み合わせ.

演算子には次のものがある.

単項演算子

"-", "!"

二項演算子

"+", "-", "*", "/", "==", "!=", ">", "<", ">=", "<=", "&&", "||"

(3) ラベル定義
<ラベル名> :

(4) goto 文
goto <ラベル名> ;

(5) 代入文
<変数> = <式> ;

(6) if 文
if (<式>) <文> [else <文>]

(7) 関数呼び出し
[<変数> =] <スレッド名>.<関数名>(<実行方法>) ;
スレッドの関数を呼び出す.
<変数>が指定されていれば, 復帰値を<変数>に格納する.
<実行方法>は, スクリプトから関数を呼び出して結果を受け取る方法を指定する.
マルチプロセス, マルチスレッドの機能を有効に利用したアプリケーションの開発を実現するため, 以下の4種類の方式が選択できるものとする.
・同期実行(DO_SYNC)
関数の処理が完了してからパケットの次の命令に移る.
・非同期実行(DO_ASYNC)
関数の処理の完了を待たずにスクリプトの次の命令に移り, 後で結果を受け取る.
・関数起動 (DO_TRIG)
関数の処理を起動し次の命令に移る. 結果は受け取らない.
・待機(DO_WAIT)
パケットは待機状態となり, 状態が変化するとその都度ユーザ定義関数が呼び出される. ユーザ定義関数が 0 以外を返すと待機状態が解除され次の命令に移る.

(8) 複文
{ <文> … }

(9) end 文
end;
パケットスクリプトを終了して自身を廃棄する.

(10) 名前空間
変数名, 関数名, ラベル名の名前空間は同じであり, これらの個々の名前は, パケット全体をスコープとする.
スレッド名とパケット名の名前空間は同じであり, システム全体をスコープとする.

(11) システム変数
パケット上にはユーザの変数宣言とは別にパケットの状態を示すシステム変数の領域が確保され, スクリプト上からの参照が可能とする.

7. 動作環境

本ツールによる開発には(3)のソフトウェアが必要であり, これをインストールし動作させるために必要な条件も含まれている. CPU の速度は推奨値を示す.
(1) ハードウェア
Pentium/230MHZ 以上の CPU
64MB のメモリ
500MB 以上のハードディスク容量
CD-ROM ドライブ
(2) OS
Windows 2000, Windows 98
(3) ソフトウェア
Borland C++Builder 5 Professional

8. 評価

(1) 開発効率
印刷機の自動運転制御プログラム (中心部分のみ) を開発し, 以前の実績と所要時間を比較してみた. 本ツールを使用することにより, 信頼性向上のためプロセスを2つに分けているにもかかわらず, 開発工数を従来の半分程度に短縮することができた.

表 1 開発工数

	従来の方式 言語 Delphi 3.1 (以前の実績)	開発したツールによる方式
プロセス数	1	2 (操作部＋制御部)
スレッド数	8	8
コーディング時間	27H	6H
デバッグ時間		5H
開発工数比較	100%	41%

(2) 実行速度
実行速度は以下のような結果となり, MMX Pentium 150MHZ において目標としていた速度である関数呼び出し 300 μsec, 関数呼び出し以外の行1行の平均実行時間 30 μsec 以内を達成できた.

表2　実行速度　　(単位：μsec)

		Windows 2000 PentiumIII 866MHZ	Windows98 MMX Pentium 150MHZ
関数呼び出しを含まないスクリプト1行の実行		0.38	2.1
関数呼び出し	同期	9	73
	非同期(最初の呼出＋2回目呼出)	20	150
	処理起動	11	67
	待機(待機している時間が無い時)	9	91
	パケットの生成と結果の受取	53	401
スレッド間の通知時間	メインスレッドとバックグラウンドで実行中のスレッド	14	345
	バックグラウンドで実行中のスレッド同士	5	155
プロセス間の通知時間		25	531
パケット生成時間		30	358

9. まとめ

スクリプトによる記述を組み合わせ，合成してシステムを記述することにより生産ラインのプログラム開発を行なうツールを開発し，生産性向上に効果があることを確認した.

今後はツールとしての使いやすさを改善し，完成度を上げること，及びスクリプトの可読性を高めるための機能追加が課題である.

10. 参加企業及び機関

(株) エヌ・エス・アール

11. 参考文献

無し