Vol. 27 No. 1 情報処理学会論文誌 Jan. 1986

ショートノート

疎結合マルチプロセッサシステムのためのシステム記述言語の設計†

馬場公光†† 甘田早苗†††

疎結合マルチプロセッサシステムでのシステムソフトウェア開発には，従来のシステム記述言語では記述しにくい点がでてくる．オペレーティングシステムのような大規模で，かつ並行に動作する部分が多く，しかも信頼性が要求されるプログラムを記述するには新しい機能が必要である．今回報告する言語は，疎結合形態のマルチプロセッサ環境でシステムソフトウェア等を記述するためのものであり，プロセス単位での並行動作，柔軟なプロセス間通信命令の提供，非決定性の指定などの特徴をもつ．この言語の概略について報告し，記述例を挙げ有効性について考察した．

1. はじめに

疎結合型マルチプロセッサシステムは，共有記憶をもたないので，各プロセッサはおのおの記憶をもち，必要に応じて通信しあう．このような環境でシステムプログラムを作成する場合，従来の共有記憶を前提とした言語では不都合な場合がでてくる．

本研究は，現在本研究室で開発中の疎結合型マルチプロセッサシステム (ADMS)*[1] 上で，システムプログラム等を記述するための言語を開発しようとするものであり，プロセス間通信，並行処理，非決定性の指定などの特徴をもっている．

2. 設計方針

この言語には，疎結合マルチプロセッサ環境で使用する，システム記述に用いる，という二つの特色がある．この章では採り入れた機能について述べる．

2.1 プロセスの記述

後に述べるように，プロセス単位での並行処理を行えるようにするため，プログラム上でプロセスを記述する．オペレーティングシステムのようなシステムソフトウェアは並行に動作する部分が多く，並行性が強く求められる．また，このとき並行に動作するプロセスの機能の把握や，これらプロセス間の相互作用がプログラム上で明確に記述されなければならない．プロセスのこのような記述により上記の問題が解決され，さらに一つのプログラムをいくつかのプロセスに分けることによりモジュール化が促進され，ソフトウェアの生産性が向上する．

2.2 並行性の指定

プロセス単位で並行動作を指定できるようにする．これは cobegin 文で行う．この方式にすると，並行動作の区間が cobegin ～ end で明示的にくくられるので，プロセスの並行動作の把握が容易になる．

2.3 非決定性の指定

非同期に並行動作しているプロセス間での通信・同期の場合のような，非決定的な制御の流れが表現できるようにする．Ada の select 文を基本にして，ADMS 用に改造したものを提供する．

2.4 プロセス間通信[2)]

柔軟性のあるプロセス間通信を表現するため，モニタによる方法ではなくメッセージの伝達によるプロセス間通信方式を採用した．この方式により，相手プロセスが同一プロセッサ（すなわち同一メモリ）上にあるなしにかかわらず，同じ方式で通信を行うことができる．疎結合型である ADMS 上でのオペレーティングシステムは，マルチプロセッサシステム全体にわたる負荷分散を行うことができるので，個々のプロセッサの負荷の程度によりプロセスを他プロセッサに移動することがあるが，この通信方式であれば問題はない．ADMS ではプロセス間通信のための道具として，メッセージとポートの二つのオブジェクトを用いる．

† On a Design of a Programming Language for Loosely Coupled Multi-Processor Systems by MASAMITSU BABA (Matsushita Communication Industrial Co., Ltd.) and SANAE AMADA (Institute of Information Science and Electronics, University of Tukuba).

†† 松下通信工業(株)

††† 筑波大学電子・情報工学系

* Advanced Distributed Multi-processor System

通信しあう二つのプロセスはそれぞれ，データなどをもったメッセージをポートに対して送る．ポートは待ち行列をもち，ここで出会ったメッセージはデータなどの受け渡しを行ったあとそれぞれが発行されたプロセスにもどる．この一連の動作で通信が行われる．

このプロセス間通信方式は，媒体の指定により通信相手を識別する．実際の記述ではプロセスのもつエントリの区別による．エントリを区別することが，プロセス間通信の媒体となるメッセージ，ポートの各オブジェクトを指定することになり，最終的には相手プロセスを指定することになる（プログラム１参照）．この方法では相手を直接指定していないかわりに自分のエントリと相手のエントリをあらかじめリンクしておかなければならない（connect 文．init は初期化指定部）．これは，通信データの型チェックなど静的なチェックに有効であり，（現時点では実現はしていないが）通信相手の変更（リンクを実行文とする），分割コンパイル時のプロセス間通信のプロセス同士のリンクのしやすさ，などプロセス間の“疎な結合関係”の表現に便利である．

プロセス間の通信には，いろいろな通信形態があるので，３種類のプロセス間通信命令を用意する．

2.5 そ の 他

この言語は基本的には Pascal をベースにして必要な機能を追加する形で構成されている．

3. 文法の概略

3.1 並列性の指定

cobegin 文によって指定する．

cobegin pi( ); …; pn( ) end

制限 ○pi～pn はおのおのプロセス名．

○( )内はデータの受け渡しに使う．

○pi～pn の記述の順番は任意である．

3.2 非決定性の指定

select 文によって指定する（**図１**参照）．

文は空文も許す．guard 中の入力文は，ただちに実行可能なときにのみ実行される．そののち－＞以降があれば実行される．

3.3 プロセス間通信

この言語で実現するプロセス間通信は３種ある．

（１） 同期通信 send/receive

send と receive の各命令がマッチすることにより通信が成立する．どちらかが先に発行された場合には，もう一方が発行されるまでその命令は待たされる．

（２） 置き去り通信 put

これは前出の receive と対応するが，receive が発行されていなくても，次の文を実行することができる．並列処理を行うプロセス間には必要な機能である．

（３） 返信付同期通信 ecall/accept

これは Ada の accept 文に似た働きをするが，引数の指定の仕方と do～end 中の文により同期信号と実際のデータの流れや順序が表現できる．

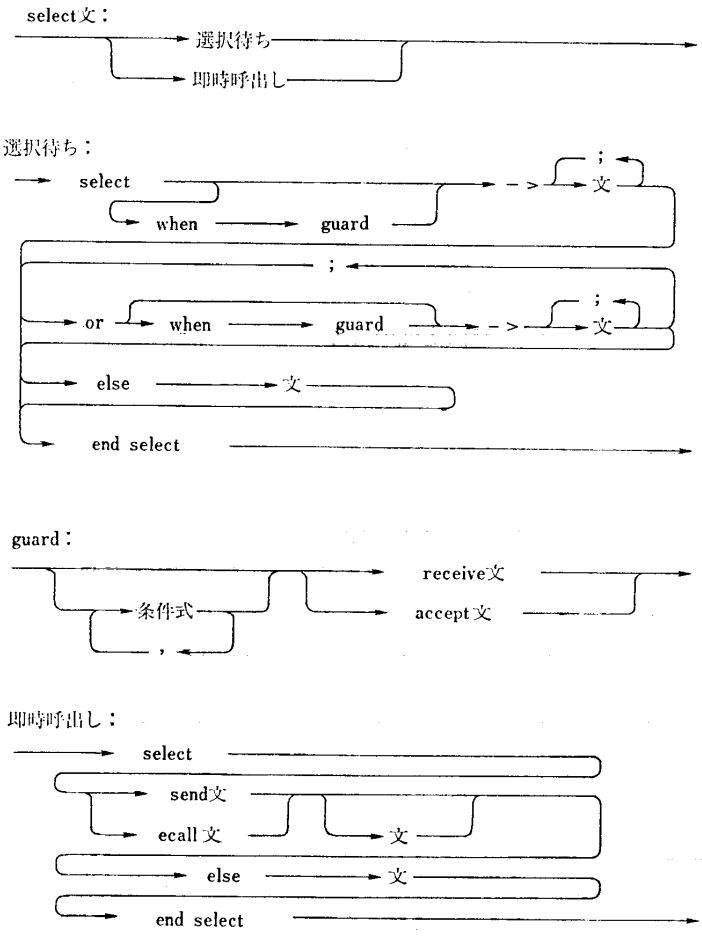

図 1 Select 文
Fig. 1 Select statement.

4. 記述例とその評価

プログラム１は三つのプロセスがまったく並行に動作するプログラム例であり，producer，consumer，buffer の三つのプロセス

```
program bounded_buffer( );
  const  N=20;

  producer=process();
    var data:integer;
    entry out(integer);
    begin
      while true do
      begin
        "produce data";
        send(out,data)
      end
    end {producer}

  consumer=process();
    var data:integer;
    entry in(integer);
    begin
      while true do
      begin
        ecall(in,data);
        "consume data"
      end
    end {consumer}

  buffer=process();
    var lastpointer,count,data,y:integer;
        buf:array[0..N-1]of integer;
    entry in(integer),out(integer);
    begin
      lastpointer:=0 ;count:=0;
      while true do
      select
        when count<>N,receive(in,data)
            -> buf[lastpointer]:=data;
               count:=count+1;
               lastpointer:=(lastpointer+1)mod N
        or when count<>0,accept(out,data)
                        do
                          y:=(N+lastpointer-count) mod N;
                          data:=buf[y]; count:=count-1
                        end
            ->
      end select
    end {buffer}

  init connect producer.out>buffer.in,
               buffer.out>consumer.in;
  begin
    cobegin buffer();consumer();producer() end
  end
```

プログラム 1 bounded_buffer
Program 1 bounded_buffer.

から成っている．buffer はラウンドロビンバッファを管理しており，producer と consumer は，これに対して書き込み，読み出しを実現するためにプロセス間通信を実行する．

このプログラムでは，buffer プロセスが producer/consumer の二つのプロセスからまったく独立している．また，buffer プロセス中のバッファ(データ構造)の操作は buffer プロセスの中に封じ込められているので，データ構造の隠蔽が行える．また三つのプロセスの動作はメインルーチンの中にまとめられているので，プログラム全体の流れの把握がしやすい．

5. むすび

これまでに述べたように，この言語の特徴はプロセス間通信と並行性の指定と非決定性の指定ということができる．これらの機能により，オペレーティングシステムのような，信頼性の要求される，並行に動作するいくつかのプロセスから成る大きなプログラムの作成が容易になる．

参考文献

1) 甘田他：分散処理用計算機のシステムアーキテクチャ，情報処理学会計算機アーキテクチャ研究会資料，48-3 (1983)．
2) 佐藤 豊：疎結合マルチプロセサのためのプロセス間通信方式，情報処理学会第 28 回全国大会講演論文集，pp. 303-304 (1984)．

(昭和 60 年 2 月 8 日受付)
(昭和 60 年 7 月 18 日採録)