ＯＳ／ｏにおける並列処理記述に関するConcurrent Programming方式の実験

4V-13

周力革，中川正樹，高橋延匡
東京農工大学工学部数理情報工学科

１．はじめに

当研究室では１９８０年にプロジェクトＰＩＥを発足し，並列処理を目的としたマルチ・プロセッサの研究、開発を行なっている。この研究用計算機システムは以下の研究をするために必要である。

（１）パターン認識，特にオンライン手書き文字認識，人工知能の研究
（２）日本語情報処理
（３）ワークステーション向きオペレーテング・システムの研究

パターン認識や人工知能の現実的な問題を処理するにはマルチ・プロセッサコンピュータは魅力的である。図１に示すようなマルチ・プロセッサコンピュータを研究対象にしている。このシステムは次の特徴を持っている。

（１）Ｉ／Ｏ、プロセッサ、メモリ資源を管理するために，一つの管理プロセッサを設ける。つまり，このプロセッサにＯＳは常駐する。

（２）すべてのプロセッサは共通バスを介して共用メモリをアクセスできる。プロセッサ間の競合はハードウエアで解決される。

（３）各々のプロセッサはローカルメモリ有する。共通バスから内部メモリへＤＭＡでデータを転送することができる。

マルチ・プロセッサは複数のプロセッサが存在するので，同じジョブを処理するために，お互いに同期したり排除したりすることを考慮しなければならない。われわれはこれらの問題を解決するのにオペレーティング・システムだけではなくて言語拡張の必要性も検討している。

マルチ・プロセッサ構成のＯＳ／ｏ[1]を実現する予備研究として，プロセス記述、同期、相互排除のためにＣ言語にプロセス、モニタを導入した。そして，これらの拡張機能をＯＳ／ｏでサポートする最小の環境を実現して，Concurrent programmingのシミュレーションを行なった。

２．並列プロセスの記述

プロセスはオペレーティング・システムがスケジューリングする単位であり，同時に実行が可能である。Ｃ言語を用いてConcurrent programmingをするときは関数をプロセスにする。

＜プロセス定義＞：：＝＜プロセス宣言＞＜プロセス本体＞
＜プロセス宣言＞：：＝ｐｒｏｃｅｓｓ
＜プロセス本体＞：：＝＜関数定義＞

システムはプロセスを生成する際に，現在実行しているプロセスと関係なく，新しいスタック領域を割り付ける。実際のプロセッサの数にかかわらず，任意個のプロセスを生成できる。

３．プロセスの同期とモニタ

Concurrent programming では並列プロセスの同期問題は重要である。共用メモリのシステムではプロセス間共用変数によって同期を行なうことが可能である。そして，その記述のためにモニタ[6]を定義する。

```
monitor モニタ名　{
        永久変数の定義及び初期化；
        condition 条件変数；

        op1 (引数リスト)
        引数の宣言；
        {
                code
        }
        op2(引数リスト)
        引数の宣言;
        {
          code
        }
        . . .
        opN(引数リスト)
        引数の宣言;
        {
            code
        }
}
```

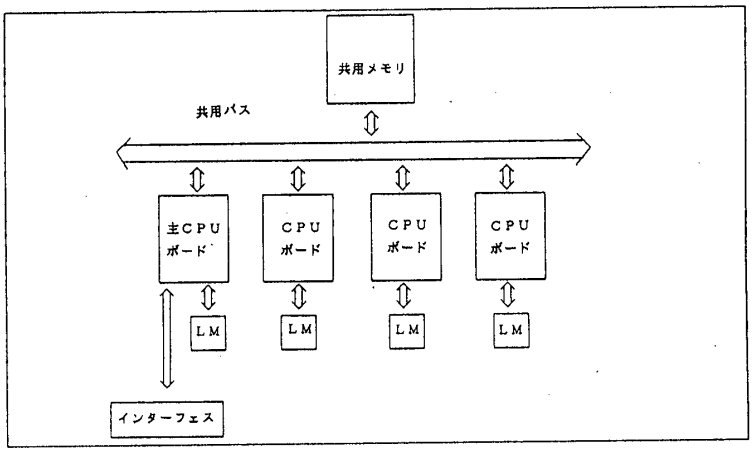

図１．マルチ・プロセッサの構成

An Experiment of Concurrent Programming for OS/omicron
Lige ZHOU, Masaki NAKAGAWA, Nobumasa TAKAHASHI
Tokyo University of Agriculture and Technology

永久変数は共用変数とその状態変数からなる。モニタの中に定義されている関数は共用変数に対してオペレーションをするものである。プロセスは共用変数をアクセスする場合に，モニタのオペレーションを使ってアクセスすることになる。多数のプロセスは同時に共用変数をアクセスしようとしても，一つのプロセスしかモニタには入れない。プロセスはモニタに入った後に，共用変数の状態によっては続いて実行できないこともある。ある情況によってある条件変数で待つ。任意個のモニタを定義することができる。

4．プリミティブ

プリミティブには起動プリミティブ，条件待ちプリミティブ，条件実行プリミティブがある。

```
cobegin(number, process1, size1, argv1, process2, size2, argv2,...)
wait(条件変数), signal(条件変数)
```

number, process, size, argvはそれぞれ起動プロセス数（ワード型），プロセス名及びプロセスのローカルデータサイズとプロセスへのパラメータである。cobeginを実行すると，スーパーバイザをコールしてプロセス名によって指定されるプロセスを生成し，メモリ領域を割り付け，パラメータを渡して起動する。waitを実行すると,スーパーバイザをコールして,このプロセスのＰＣＢが退避され，引数で指定された条件変数の待ち行列に入る。signalを実行すると，現在のプロセスはサスペンドされて,この条件変数の待ち行列からプロセスを起動する。

5．シミュレーション

5．1スーパーバイザの基本構成

プロセスを管理するために，プロセスの状態，同期信号，メモリなど資源の情報を保存しなければならない。プロセスを生成する際に，プロセスごとにこの情報の記憶領域を与え，これをＰＣＢという。ＰＣＢの集合はプロセスの管理表を構成する。これはスーパーバイザの一部としてメモリに常駐する。一つのプリミティブcobeginによって生成されたプロセスは一つのプロセス族を構成する。プロセス族を生成したプロセスは親プロセスという。プロセスを生成しないプロセスは無族プロセスという。親プロセスはかれのプロセス族が終了しなければ実行を再開できない。プロセスの世代関係を管理するのにプロセス族の管理表を作成する。

プロセスの生成、終了によって，その記憶領域を生成したり，回復したりすることを行わなければならない。また、プロセスの実行を中止したり再開したりするために，配置情報も必要である。一つのローカルメモリに対してマルチ・スタック管理表が必要である。複数個のローカルメモリに対して,プロセスはどのメモリにあるかを示すために，ローカルメモリ管理表も必要である。

5．2メモリレイアウト

ＯＳ／ｏの実行環境[2]~[5]は手続き領域、大域データ領域、ヒープ、スタック領域からなる。マルチ・プロセッサの場合には，すべてのプロセスは大域データ領域を共用する。各々のプロセスはヒープ、スタックを固有に持っている。現在はプロセッサが4つあることを仮定すると，共用メモリに大域データ領域を割り付けて，ローカルメモリにヒープ、スタック領域を割り付ける。図２に示すような環境でシミュレーションを行った。

6．おわりに

マルチプロセス、モニタを導入してＣ言語を用いてコンカレントプログラミングを行なった。特に，モニタは共用領域の相互排除に対して有効であるし，他の方式に比べてプログラミングしやすい。

パイプライン処理，木構造の生成、探索，数値計算などの実験によると，コンカレントプログラミングはマルチプロセッサによる並列処理を記述するのに便利である。

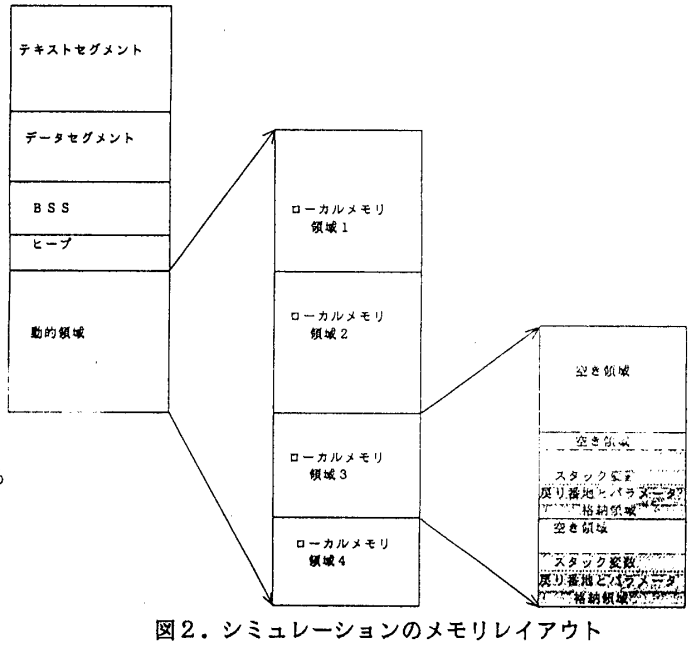

図２．シミュレーションのメモリレイアウト

参考文献

1．高橋延匡，他，“ＭＣ６８０００用ＯＳ：ＯＳ／ｏの開発”，情報処理学会計算機システムの制御と評価，１９８３．１２

2．中川正樹，他，“ＭＣ６８００ユニ&マルチ・プロセッサ・システム用システム記述言語Ｃ処理系の開発”，情報処理学会計算機システムの制御と評価，１９８３．１２

3．藤森英明，他，“ＭＣ６８００マルチプロセッサシステム開発用言語Ｃコンパイラの基本設計”，情報処理学会第２８全国大会資料，１９８４

4．篠田佳博，他，“ＭＣ６８００マルチプロセッサシステム開発用言語Ｃコンパイラの実現”，情報処理学会第２８全国大会資料，１９８４

5．武山潤一郎，他，“ＯＳ／omicron第一版におけるマルチタスクの実現”，情報処理学会第２８全国大会資料，１９８４

6．BRINCH HANSEN,P.,“The Architecture of Concurrent Programs”,PRENTICE-HALL,INC., Englewood Cliffs, NJ, 1977