EMPS：多重プロセッサシステム*

横井　俊夫**　笹岡　靖正***　内藤　富久****　淵　　一博**

Abstract

Several problems on the operating system of multiprocessor systems, and some general solutions of these problems are discussed.

The EMPS (ETL Multi-Processor System) developed by the authors is reported. The basic approach to multiprocessor systems, the scheme of the inter-cpu communication (control), overviews of the EMPS (hardware and software system), the method of assignment of cpu's to processes, the structure of the Process Control program, and a general discussion on interlocks from the viewpoint of preventing system deadlocks are given. These methods are successfully implemented in the EMPS.

1. はじめに

本論文は，一般の商用計算機に，多重プロセッシング用の最小限の機能を付加したものを用いて，多重プロセッサシステムを構成する際の操作システム（制御プログラム）の問題点の指摘と，有効なる諸方式の提案を目的としたものである．

信頼性・可用性・処理能力などの増大を目的として，多重プロセッサシステムが研究・試作されてから，すでに久しく，アメリカにおいては実用の段階にはいっている．わが国においても実用化にさしかかったところである．近年，計算機処理速度の向上への強い要求から，新しい方式による計算機の構成，すなわち，並列処理の機能に重点をおいた計算機の研究が盛んになってきている．このような動きの中で，多重プロセッサシステムも，実用化されたものがどちらかといえば，信頼性・可用性の増大に重点がおかれていたものが，処理能力の増大という面が重視されだしてきた．本格的な並列処理計算機の実現までの過渡的なものとして，また，最も一般性のある並列処理計算機システムとして，多重プロセッサシステムが見直されてきた観がある．さらに，大規模な実時間処理システムの普及に伴って，信頼性・可用性の向上の面からの価値も増しつつある段階である．

以上のような要請に対して，多重プロセッサシステムを普及させるためには，いままでの実用化された多くのもののように，特殊なハードウェアを必要とする計算機ではなく，一般の商用計算機に若干の多重プロセッシング用のハードウェアを加えたものを用いて，適切な操作システム（制御プログラム）を構成することにより，有効な多重プロセッシングが実現できなければならない．このような観点に立って，われわれは多重プロセッサシステム，EMPS (ETL Multi-Processor System) を研究・試作した．

以下において，EMPS 自体の概観，試作の基本方針となったいくつかの方式，および一般的考察を含めて述べることにする．EMPS で使用した計算機は，以下で述べるように多重プロセッシング用としては，最小限の機能しかもたないもので，操作システムの構成には多分に苦慮したところがあるが，ここで述べる諸方式は，どのような多重プロセッサシステムの操作システムにも共通な，一般なものである，EMPS 自体は，人的制限などから，すでに長期間稼動中の ETSS[1] を多重プロセッサ用に改良・拡張したもので，すでに昭和44年9月に基本部分は完成し，以後，性能測定と，それに基づく若干の改善が行なわれ，現在に至っている．

2. 基本方針

* EMPS: Multiprocessor System, by Toshio Yokoi (Electronic Computer Devision, Electrotechnical Laboratory), Yasumasa Sasaoka (Hitachi Ltd.), Tomihisa Naito (Japan Systems Creation Inc.), Kazuhiro Fuchi (Pattern Information Devision, Electrotechnical Laboratory).

** 電子技術総合研究所

*** 日立製作所

**** 日本システムクリエーション

この節では，EMPSを試作する上での基本的な考え方を述べる．しかし，その前に，多重プロセッサシステムを明解に議論するために，一般の1台の処理装置による計算機システムの構造を整理してみる．

通常の処理装置は，少なくとも2つの実行モードからなっている．1つは，割込みの受付けと解析を行なう．したがって，割込み禁止の Master (Supervisor) モードと，もう1つは，割込みに基づく処理や利用者のプログラムが走しる，割込み許容の Slave (User) モードである．操作システムのほうからいえば，Master モードにおかれる制御プログラムは，割込み基づいて，process (task) の状態を管理し，割込み信号を適切な process への event 信号に変換する．すなわち，Slave モードの多重化を行ない，一般に Process 制御部 (Task 管理) などと呼ばれている．一方，Slave モードで実行されるプログラムは，システムプログラムも，利用者のプログラムも，Process 制御部により作り出された process として表現される．

以上のような2つの実行モードの役割から，処理装置を Master モードと Slave モードの2つの計算機からなるものと考えるのである．すなわち，Slave モードの計算機が動いているときは，Master モードの計算機は，割込み信号に対する Wait 命令を実行していると考えるのである．割込み信号がくると，Master モードの計算機は，Slave モードの計算機をとめ，process に対する処理を行なう．処理装置の別の見方として，あくまで1台の計算機があり，割込みにより状態が変わるという見方がある．しかし，多重プロセッサシステムを考える際には，本論文のような見方のほうがより適切である．そして，この考え方は，実行モードが3つ以上の場合にもそのまま拡張される．各実行モードに対応した計算機の複合体として，処理装置を考えればよい．

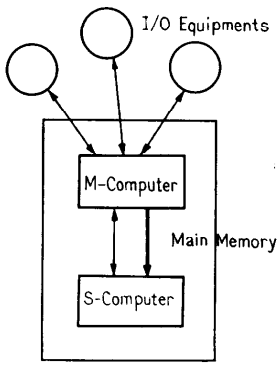


Fig. 1 The model of a signal processor system

この考え方に基づいて，1台の処理装置からなる一般の計算機システムを図示すると **Fig. 1** のようになる．入出力装置は，チャネルをも含めて単純化して表わしてある．M-計算機は，Master モードの計算機，S-計算機は，Slave モードの計算機を表わす．2つの計算機を囲む実線のわくは，主記憶を表わし，2つの計算機がアクセスできる主記憶が同じものであることを表わしている．入出力装置と主記憶の関係は議論を簡単にするために省いてある．入出力装置および2つの計算機間の実線の矢印は，それぞれ間で割込み信号，すなわち，event 信号により，相互に連絡可能なことを示している．M-計算機から S-計算機への太線の矢印は，M-計算機が S-計算機を制御することができることを表わす．すなわち，S-計算機をとめたり，動かしたり，それに伴い S-計算機の状態レジスタ（プログラムカウンタなど）の内容を読み出したり，書き換えたりできることを表わしている．Fig. 1 のような1台の処理装置による計算機システムから，多重プロセ

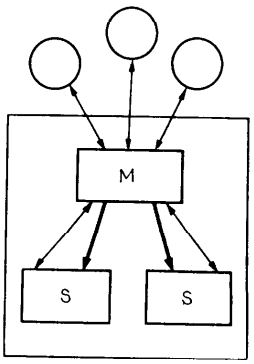


Fig. 2 The model of a multiprocessor system with one M-computer

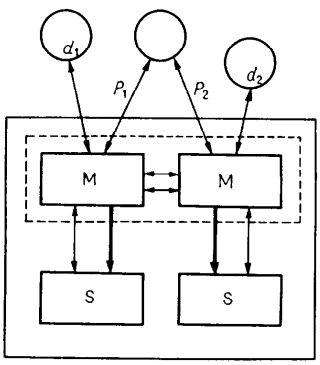


Fig. 3 The model of an ordinary multiprocessor system

ッサシステムに移行するとき，最も自然な方向は，**Fig. 2** のような多重プロセッサシステムを構成することである[2]．Fig. 2 の操作システムは，Fig. 1 の操作システムの M-計算機のプログラムを若干拡張すればよいだけである．つまり，ready 状態にある process のうち，優先順位の高いものから2つを選び，それぞれを2つの S-計算機に割り当てればよい．これ以外の変更は何ら必要としないで，Fig. 1 から Fig. 2 の計算機システムへ移行することができる．Fig. 2 では S-計算機を2台としたが，3台以上としても同様の議論が成り立つ．なお以下においても，すべて2台ということで議論を進めるが，3台以上の場合にもすべて同様に考えればよい．論旨を簡明にするため，特にことわらない限り，2台ということで進める．

一方，EMPS で対象とした，また一般の多重プロセッサシステム（主記憶共有）は，**Fig. 3** に示すものである．M-計算機間の2つの矢印は，後で述べるダイレクト・コントロールと Test & Set 命令という2つの連絡手段があることを示している．なお，d_1, d_2, p_1, p_2 などは，5節で説明に用いるものである．Fig. 3 のような計算機システムを多重プロセッサシステムとして構成する最も自然な方法は，Fig. 3 を Fig. 2 のシステムのように構成することである．つまり，Fig. 3 で点線で囲んだ部分を，S-計算機から見たとき1台の M-計算機であるかのように模擬すること，そのように M-計算機の制御プログラムを構成することである．このような多重プロセッサシステムは，通常，対称型の多重プロセッサシステムと呼ばれるもので，負荷のバランスが容易にとれること，制御プログラムが簡明になることなどの利点が指摘されている．

以上のような考察に立って，EMPS を試作する上での基本方針は，Fig. 3 の計算機システムを徹底して Fig. 2 の計算機システムに近づけることである．こうすることにより，Fig. 1 の計算機システム用に構成された ETSS を，簡単に，能率のよい多重プロセッサシステムとして，Fig. 3 の計算機システムの上に構成することができる．また，処理装置の故障などから，1台の処理装置を切り離して，Fig. 3 から Fig. 1 に移る場合を考えると，この際の処置をつかさどる制御プログラムの部分は，非常にわずかとなり，その部分も論理的に簡明なものとして構成することができる．

次に EMPS の概観を示すために，ハードウェアシステムとソフトウェアシステムの両面から簡単な説明を行なう．

3. EMPS の概観

（1） ハードウェアシステム[9] EMPS は H-8400 を多重プロセッシング用に改造した H-8410 2台からなる．（主記憶） 4つのバンクからなり，256 KB の共用メモリと 4 KB ずつの専用メモリからなっている．処理装置と各バンクは，メモリスイッチを介して接続されている．（処理装置間の連絡） Test & Set 命令のほかにダイレクト・コントロール機構という割込みによる同期化の手段がある．

（2） ソフトウェアシステム EMPS 全体の機能は，多重プロセッシングを行なう TSS である．TSS としては ETSS と同じである．M-計算機のプログラムを，ETSS と同じく Process 制御部（Process Control Module）と呼ぶ．これがシステムの中核をなす．ほかのシステムプログラムおよび利用者のプログラムは，process として S-計算機で実行される．いずれの S-計算機で実行されるかは，process としては関知しない．すべてのシステムプログラムは，システムに1つである．以下において，先の基本方針に基づき，Process 制御部を構成する上での問題と，その解決の方式を中心として議論を進める．

4. 処理装置間の相互制御

Fig. 3 のシステムを Fig. 2 のシステムと同価にするためには，割込み信号や Test & Set 命令を用いて，必要に応じてほかの処理装置の状態を変える手段が必要である．ここでは，ほかの処理装置の状態を変えるための連絡の手段を，相互制御の方式として述べることにする．

まず，Fig. 2 のシステムの M-計算機は，次の2つの特性をもつ．第1は，すべての S-計算機およびすべての入出力装置からの割込み付号は，この1台の M-計算機に向けられている．そして，多くの割込みが同時に生じてもそれらを逐次受け付ける機能を M-計算機はもっている．第2は，この1台の M-計算機が，すべての S-計算機を制御できる．すなわち，すべての S-計算機を多重化できる機能を，この M-計算機はもっている．一方，Fig. 3 のシステムは，上記の Fig. 2 の2つの特性に対比して，もし相互に連絡手段がないとすると，次のような性質をもつ．第1は，各 S-計算機および各入出力装置からの割込み付号は，それぞれが接続されている M-計算機に向けられる．したがって，2台の M-計算機は，互いに独立して動く

ものであるから，すべての割込み付号を遂次処理する機能がないことになる．第2は，各 S-計算機に対する制御権は，それぞれの M-計算機にあるため，一方の M-計算機から，他方の M-計算機に属する S-計算機を制御することができない．

以上のような2つのシステムの性質から，Fig. 3 のシステムを Fig. 2 のシステムと同価にするためには，次の2つの機能が Fig. 3 のシステムに必要である．第1に，2つの M-計算機を遂次的に動かすための相互連絡の手段が必要である．第2に，一方の M-計算機から他方の M-計算機への割込みによる連絡手段．ただし，この割込みは，他方の M-計算機が受け付けうるほかの装置からの割込みのすべてに優先して，少なくとも，受け付けた他の装置からの割込みの具体的な処理が始まる以前に，他方の M-計算機が受け付けうるものであることが重要である．この割込み信号によって，ほかの M-計算機からの要求どおりに，S-計算機の状態を変更することができる．

次に，上記の2つの機能を，Test & Set とダイレクトコントロール機構という割込み機能を用いて，いかに実現するかを述べることにする．議論を一般的にするために，Test & Set 命令の機能もそのほかの割込みの機能も，すべての相互連絡（同期化）の手段を，event 変数とそれに対する NOTIFY 命令と WAIT 命令の機能に統一する．この2命令の機能は次のようなものである．$e_1, e_2, \ldots, e_i, \ldots, e_n$ を event 変数とすると，NOTIFY (e_i); は，e_i="1" とする．WAIT $(e_1, e_2, \ldots, e_i, \ldots, e_n)$; は，$e_1, e_2, \ldots, e_n$ がすべて "0" のときはこの命令のところでとどまり，先へは進まない．いずれか一つ，たとえば e_i が "1" となると，e_i="0" として，次の命令の実行に移る．また，どの event 変数により Wait 状態が解かれたかは，なんらかの方法で知らされるものとする．2つ以上の event 変数が同時に "1" になった場合は WAIT 命令のオペランドのうち，先に書かれたものほど優先順位が高いものとする．この2命令を用いて Process 制御部を PL/I に似せた言語で表わすと，**Fig. 4** のようになる．ただし，このプログラムは，処理装置間の相互制御を中心にして，そのほかは省略してある．なお，図の右側の番号は説明の便宜上つけたものである．Fig. 4 の (1) は NOTIFY あるいは WAIT 命令の対象となる event 変数のみを宣言したもので，t-event は，2つの M-計算機に共通で，各 M-計算機において NOTIFY, WAIT 命令の対象となる．これは，

```
process_control_module: PROC; /*M-computer*/
  DECLARE t_event EVENT_VARIABLE INT ("1"),
          m_event EVENT_VARIABLE,
          y_event EVENT_VARIABLE, ·····(1)
          i_event EVENT_VARIABLE,
          e_event EVENT_VARIABLE;
    start: pre-processing.
     wait: WAIT (t_event, m_event);·····(2)
          IF m_event occured THEN GO TO receive_1;
          main-processing-1;
          IF control is necessary THEN GO TO send;
          main-processing-2;
          NOTIFY (t_event);
     exit: post-processing;
          WAIT (m_event, e_event, i_event); ·····(3)
          IF m_event occured THEN GO TO receive_2;
          GO TO start;
     send: NOTIFY (y_event);
                ⋮
          GO TO exit; ·····(4)
receive_1: modify s-computer's states;
          NOTIFY (t_event);
          IF e_event occured THEN GO TO wait; ·····(5)
          GO TO exit;
receive_2: modify s-computer's states;
          NOTIFY (t_event);
          GO TO exit;
    END;
```

Fig. 4 Inter-cpu control scheme

Test & Set 命令と同様の機能を表わしたものである．m-event および y-event は，M-計算機間での割込みによる連絡に用いられるもので，M-計算機は m-event に対しては，WAIT 命令，y-event に対しては NOTIFY 命令を実行する．したがって，一方の M-計算機で，m-event, y-event であるものが，他方の M-計算機では同一の変数が，逆に，y-event, m-event として扱われる．Fig. 4 では簡単のため，この間の表現は省略してある．i-event, e-event は，そのほかの割込みを受けつけるためのもので，i-event は S-計算機からの各種の内部割込みに対応し，e-event は各種の入出力装置からの外部割込みに対応したもので，簡単のために，ひとまとめにして表現したものである．Fig. 3 の構成からもわかるように i-event, e-event は，各 M-計算機によって，異なる変数である．Fig. 4 で一番問題になるのは，実際の処理装置には，(2) のような WAIT 命令を実行しうる機能のがないことである．そこで EMPS では Test & Set 命令を含むループの中で，他処理装置からの割込みフラグを検査し，信号の有無を確かめるようにして，この (2) の機能を模擬している．(3) は先にも説明したように実際にはハードウェアの機能である．通常は (3) に至るまでに，必ず，t-event に対する NOTIFY 命令が実行され，ほかの M-計算機が

main-processing を行ないうるようにしているが，(4) の場合が例外である．この場合は，ほかの M-計算機に，S-計算機の状態を変える (prcoss の入れ換えを行なう) ように y-event に対して NOTIFY 命令を実行した後である．そして，この NOTIFY 命令によって信号を受け取った M-計算機 ((2) の WAIT 命令により) が優先して main-prcessing を行なえるようになっている．receive-1 の部分は，ほかの割込み信号を受けつけ，しかし，まだ処理をしていない前に (2) によって，ほかの M-計算機からの信号を受け取った場合である．この場合には，(5) に示したように外部割込みか，内部割込みかによって処理が異なる．S-計算機の状態を変えるということは，S-計算機で走っていた process をほかの process と入れ換えるということであるから，内部割込みのときは，その情報を入れ換えられた process の状態語の中に残し，実際の処理は次にその process が走しるまで延ばす．一方，外部割込みの場合は，process の入れ換えには，なんら無関係であるから，(2) の時点にもどる．普通 M-計算機間で，Fig. 4 のような連絡が必要になるのは次の3つの場合である．

① 自分の S-計算機で実行中の process からの要求で，他方の S-計算機で実行中の process に作用が及ぶ場合 (process の停止など)．

② 優先順位に基づく，ほかの S-計算機の先取り．

③ 異常事態．

したがって Fig. 4 では省略したが，より細かな情報も伝える必要がある．このときは Fig. 4 に mail box[4] を設ければよい．Fig. 3 を Fig. 2 と同価にするためには，Fig. 4 の (2) の WAIT 命令の機能が重要であることは，すでに述べたが，この WAIT 命令の機能がなくとも，Fig. 3 のM-計算機の上に，もう一つ MM-計算機 (master of master) をおいて，**Fig. 5** あるいは **Fig. 6** のようにすれば，同価にすることが可能である．特に，Fig. 6 の場合は，各処理装置に master モードの上にもう一つの割込みレベルを設け，処理装置からの割込みはその割込みレベルで受けられるようにした場合である．

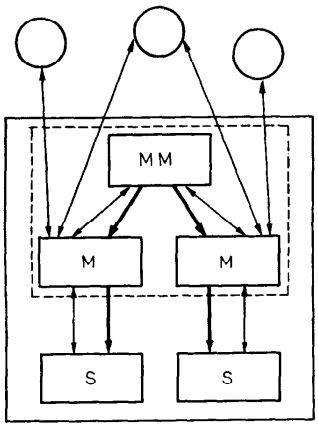


Fig. 5 The model of a multiprocessor system with one MM-computer

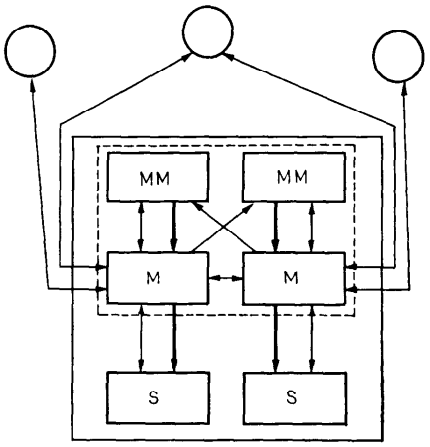


Fig. 6 The model of a multiprocessor system with two MM-Computer

なお，Fig. 4 のプログラムでは，一方の M-計算機が main-processing 中に故障し停止してしまったときには，システム全体が停止する．これを防ぐためには，タイマをセットし，(2) の WAIT 命令で，タイマの割込みも待つようにすればよい．EMPS では，このようなタイマの機能がないため，先の (2) の WAIT 命令に対応する Test & Set 命令のループの中で，ループをまわった回数をカウントすることで代用している．

5. 実行処理装置の指定とその処理

多くの process は，実行される S-計算機には制限を必要としない．しかし Fig. 3 の d_1, d_2 のように処理装置 (M-計算機とそれが制御する S-計算機を一体としたもの) に固有の機能 (装置) で，他の処理装置からは直接アクセスできない機能がある場合，また，p_1 と p_2 のようにいずれの処理装置からも一応アクセスできるが，process としては，2つの経路を区別して機能を使用したい場合，いずれの場合にしろ，なんらかの実行処理装置を指定する手段が必要になる．

以下では，まず処理装置を指定する方法と，次に処理装置を指定した process の一般的な処置法について述べる．

実行処理装置の指定には，暗示的な方法と明示的な方法の2つが考えられる．

（1） **暗示法**　process は，少なくとも Fig. 3 のような状況にある機能の使用は，すべて Process 制御部（M-計算機）を介する．この際すべての機能に固有の名前をつける．Process 制御部は，各名前とその機能がいずれの処理装置に属するかの一覧表をもっている．この表を介することにより，Process 制御部が機能の使用要求を処理できる処理装置を決定する．もし自分で処理できぬ場合は，その機能にアクセスできる処理装置に，その要求の処理を依頼する．依頼方法であるが，要求それ自体を頼むよりは，要求を出した process 全体の実行を頼む方が論理的に整理しやすい．

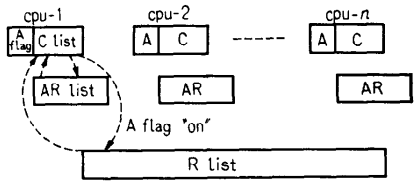
（2） **明示法**　Process 制御部に，実行処理装置を指定したり解除したりするための入口（entry）を設ける．process は，各機能がどの処理装置で使用可能かを判断し，Process 制御部に処理装置の指定を指示する．次にその機能を使用し，終わった時点で指定解除を Process 制御部に知らせる．

以上でいう機能の使用要求とは，入出力命令などの実行をさし，したがって，処理装置を指定したままで，process が wait 状態になることはないとする．

暗示法は，機能と処理装置との接続関係を一覧表として，Process 制御部が一括して管理し，process としては，一切具体的な接続関係を知る必要はない．したがって，接続関係の変更に際しても，Process 制御部の一覧表を変更するだけでよい．明示法はより一般的である．しかしながら，指定と解除の2つを Process 制御部を介さなければならず，process が必要とする機能の接続関係を知っていなければならない．

EMPS では，先に述べた基本方針を徹底させるために，暗示法を採用した．なお暗示法では，指定された処理装置で，process は再び要求を Process 制御部に出さなければならない．これに関しては，EMPS では，Process 制御部が process のプログラムカウンタをもどすことにより，該当 SVC 命令が再実行される方法を用いた．次に上記の方法で実行処理装置を指定された process を，いかに Process 制御部で処理するかを述べる．

Process 制御部には，現在 S-計算機で実行中の process（状態語）をさすCリストに呼ばれるポイントと，実行可能ではあるが，S-計算機のあきを待っている process 群を優先順にならべたRリストと呼ぶものがある．Cリストは処理装置ごとにあり，Rリストはシステムに1つである．より高い優先順位の process に割り込まれると，Cリストの process はRリストに移される．process の実行処理装置の指定がない場合は，実行可能状態にある process の制御には，このCリストとRリストで充分である．処理装置の指定がある場合には，これを一律に処理するために，**Fig. 7** に示す AR リストと A フラグを各処理装置ごとに設ける．AR リストは，その処理装置を指定した process で実行待ちの状態にあるものを優先順にならべたものである．A フラグは，Cリスト上の process が処理装置の指定をしたものか否かを示す．処理装置があいたとき，新しい process をRリストと AR リストのいずれからとってくるかは優先順位による．同じ優先順位の場合は，AR リストから取り出す．処理装置の指定が行なわれると，指定の処理装置が自分の場合は，Process 制御部は，A フラグを “on” にして process の要求の処理，あるいは process の実行を続けさせる．指定された処理装置が自分でない場合は，該当する処理装置の AR リストに優先順位順に process をつなげる．このとき，相手の処理装置が A フラグが “off” で，AR リストに入れた process より，優先順位が等しいかより低い process を実行中の場合と，A フラグが “on” で，より低い優先順位の process を実行中の場合には，**4.** に述べた方法により，process の入れ換えを相手の処理装置に知らせる．そのほかの場合は，単に AR リストにつなげるだけである．Cリストの process が，より高い優先順位の process に割り込まれると，その process は，A フラグが “on” のときは AR リストへ，A フラグが “off” のときはRリストに移されて処理装置のあきを待つ．Process



A flag : Assigning flag
C list : Pointer to currently running PSW
AR list : List of PSW assigned to the cpu
R list : List of ready PSW

Fig. 7 Lists in the Process Control Module

制御部は，process からの要求の処理が終わると，また，指定解除を受けとるとAフラグを “off” にする．処理装置があき，新しい process を取り出す場合，AR リストからのときはAフラグを “on” にする．この AR リストとAフラグにより，処理装置の指定を論理的にも能率の面からもうまく処理することができる．

6. Process 制御部プログラムにおける処理装置上の相違の吸収

先の基本方針に従い，処理装置に関する台数をも含めた種々の相違を，すべて Process 制御部で吸収する．この際に，Process 制御部のプログラムは，システムに1つだけおき，各処理装置（M-計算機）が re-entrant に使用できること，および処理装置の台数が変化しても，Process 制御部のプログラムはできるだけそのまま使用できることが望ましい．このような要求を満たす方法の一つに，ハードウェアとして Prefixing[5] と称する機能を設ける方法がある．ここで述べる方法は，特別なハードウェアの機能はなく，ただ専用メモリのみを用いる解決法である．専用メモリとは，すべての処理装置が同一のアドレスを用いても，物理的には，それぞれ異なった場所を指定し，しかも，互いにほかの専用メモリを参照することができないような記憶場所のことである．処理装置の固有名，**5.** で述べた AR リストや Aフラグ，作業領域，異常時の状態の保存領域などを一括して MSW (M-compuer Status Words) と呼ぶ．MSW は，処理装置の基本的な状態を表わすもので，ほかの処理装置からも参照される．そこで，MSW は，システムに存在する処理装置の数だけ共用メモリにおかれる．**Fig. 8** に専用メモリと固有メモリにおける MSW の有様を示す．各処理装置の専用メモリの同一番地にポインタがおかれ，自分の MSW をさす．これにより，一般のプログラムと同様に，re-entrant 化の最低限の準備はできたことになる．一般の re-entrant プログラムと異なり，Process 制御部のプログラムには，ほかの処理装置の MSW を順次参照する機能が必要である．この MSW の順次参照を一つのプログラムで実現するためには，どの処理装置から見ても同じ条件になるように，MSW 群を配置しなければならない．そこで Fig. 8 に示すように，MSW をポインタを用いてリング状のリストに作る．MSW を参照する方法は，自分の MSW から出発して，終わりは，また自分の MSW に帰ってきたことで判定すればよい．この参照の方法は，どの処理装置からも，また処理装置の台数が変化しても一律に適用できる．処理装置の台数を変えるときは，対応する MSW をリストにつなげたり，はずしたりすればよく，Process 制御部のプログラム自体はなんら変更する必要はない．

以上が一般的な方法である．次に EMPS で採用した方法を述べる．EMPS の Process 制御部のプログラムは，処理装置の台数が2台か1台の場合にのみ適用できるように，制限を設けて製作した．このように最大2台までという制限を設けると，上記の方式よりもっと簡単で有効な方式が考えられる．EMPS の方法を **Fig. 9** に示す．2つの MSW へのポインタは，自分のもの相手のものという順序で，2つの処理装置の専用メモリ上で同一の番地に配置する［Fig. 9 (a)］Process 制御部のプログラムは，my→，your→ という方式で記述する．処理装置の台数が最大2台という制限をつけているので，MSW を順次参照する際

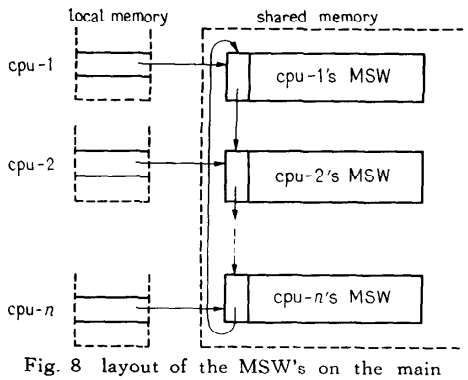


Fig. 8 layout of the MSW's on the main memory

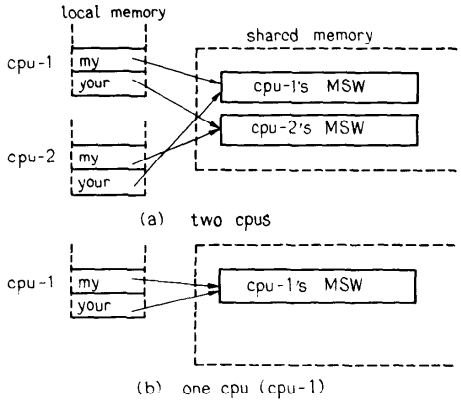


Fig. 9 MSW's layout in case of the EMPS

に終わりを判定する必要がない．したがって，先の方法より若干能率のよいプログラムが書ける．Process 制御部のプログラムを，そのまま1台の処理装置で走らす際の処置は次のようにする．たとえば，CPU-1だけを使用する場合は，Fig. 9 (a) の専用メモリの内容を Fig. 9 (b) のように書き換える．専用メモリ上の2つのポインタはともに CPU-1 の MSW をさす．CPU-2 を1台だけで使用する場合も同様の処置をとればよい．Process 制御部のプログラム自体は，何の変更も必要としない．しかし，プログラムが my→ と your→ によって書かれており，1台で使用するときはともに同一の MSW を参照するのであるから，プログラムには次のような制約が必要である．自分と相手の MSW の内容を比較していずれを取るかを判定する部分では，判定した結果，条件が同じときは，常に自分のほうを選ぶ．この制約は，Process 制御部プログラムの製作上，なんら負担とはならなかったし，能率にも影響を与えない．この EMPS の方法は，3台以上の場合にも拡張できるが，このときは先の一般的な方法に比べ，必ずしも能率がよいとはいえない．あくまで最大2台という制限の上で有効な方法である．

7. Interlock

ここでは，種々の面から研究されている interlock を，deadlock 防止の観点から体系だてて述べる．あわせて，EMPS で採用した方法についても述べることにする．

多重プログラミング，多重タスキング，多重プロセッシングなどと呼ばれる特性をもつシステムでは，多くの process (task) および job が，限られた resource を競合しながら使用する．このとき，使用中にほか process のから乱されるのを防ぐ手段が必要である．この点に関しては，Dijkstra の P-operation, V-operation や Dennis & Van Horn による LOCK, UNLK マクロや長期間の resource の使用を待合せるための ENQUE, DEQUE マクロなどがある．EMPS では，短期間の resource 使用に対しては LOCK, UNLK マクロを，長期間に対しては，すでに ETSS に設けられている SEIZE, RELEASE の機能を用いた．ここで，短期間とは，resource の使用が，process を wait 状態にするための処理時間より短いと思われるような場合である．

次に問題となるのは，process が2つ以上の resource を同時使用する際に生ずる deadlock (hang-up) と呼ばれる問題である．この点に関しては，種々の観点から，防止法や発見法について考察が行なわれているが，実際の操作システム（制御プログラム）に当てはめると個別的にすぎる．そこで，整理して若干体系だててみることにする．

一般に，独立して動くもの同志が共通に使用し，使用中はほかからの使用が禁止されるようなものは facility と呼ばれ，独占して使用するための手段が interlock と呼ばれる．facility を取りにいく動作を，ここではロックすると呼び，独立して動くものはすべて process と呼ぶことにする．実際の操作システムでは，interlock は種々の場面で階層的に表われ，いろいろの手段が用いられる．一番の根本になるのが，Fig. 1 の M-計算機が，一時には1つの割込みしか受け付けないことである．すなわち，M-計算機が次の割込み待ちの状態になるまで，ほかの割込み信号は，ハードウェアによって待たされる．そして Fig. 3 のように M-計算機が2台以上ある場合には，Test & Set 命令などによって，interlock の次の手段が設けられている．これらの完全にハードウェアによる段階を L_1 とする．次の段階は，LOCK, UNLK に対応する interlock である．この段階では，ロックできないからといって，process が wait 状態になることはない．この段階を L_2 とする．次は，ENQUE, DEQUE に対応する interlock である．ここでは，ロックできない場合は，process はその facility があくまで wait 状態になるし，ロックしたままで wait 状態になってもよい．この段階を L_3 とする．その次は，ファイル内のレコードの参照，書換えによる interlock などが続くことになる．いずれにしろ，各段階の interlock は，一段上の interlock の保証によって成り立っている．以下では，代表的な L_1, L_2, L_3 の3つのレベルについて，deadlock 防止の問題を論ずることにする．

各レベルに対応していくつもの facility があり，process は，場合によって，レベルにまたがりいくつかの facility をロックする．レベル間での deadlock を防ぐ方法は，次のような方法である．

① L_1, L_2 の間では，同一の facility を用いない．つまり，M-計算機自体を1つの process と考えると，M-計算機がロックする facility，すなわち，レベル L_1 の facility を S-計算機から使用する場合は，すべて M-計算機を介する．

② L_2 と L_3 の間では，process が wait 状態になるときは，ロックしていたレベル L_2 のすべて

の facility のロックを解く.

この2つの方法により，レベル間での deadlock は生じなくなる．EMPS でもこの方法を用いた．次に同一レベルの facility 群での deadlock の防止法について述べる．L_1 では，その性質上あまり繁雑な方法は不適当である．せいぜい facility を少数にまとめ，ロックする順番を一定に定めておくことくらいである．EMPS では，facility をまとめて一つとした，L_2 では，やはりあまりこった方法はかえってシステムの効率を落す，一方，多重プロセッサシステムの場合は，あまり杜撰な方法では，やはりシステムの効率を落とす．そこでロックする順番を一定にするとか，必要なものがすべてロックできるまで待つとかの方法がとられる[6]．EMPS では，順番を一定にする方法を若干改良したものを用いた．EMPS の方法を簡単に説明する．

L_2 のレベルにある facility の集合を，$F=\{f_1, f_2, \ldots, f_n\}$ とする．process が f_i をロックして使用するときには，あらかじめ f_j をロックしていなければならない場合には，$f_i > f_j$ と順序づける．このようにして，F の全要素に順序づけを行なったとき，もし $f_{i0} > f_{i1} > \ldots > f_{in} > f_{i0}$ なる順序づけが生じたときには，$\{f_{i0}, f_{i1}, \ldots, f_{in}\}$ をまとめて，1つの facility f_i' とする．このようにして，F から F' を得る．したがって F' は，等しい要素をもたない半順序集合となる．process が facility をロックすると，process の優先順位はロックした facility に対応して高くする．$f_i < f_j$ のときは，$p_r(f_i) < p_r(f_j)$ である．ただし，$p_r(f_i)$ とは，f_i をロックしたときの process の優先順位である．このようにすれば，F' の性質上 deadlock は生じない．

L_3 では，いろいろとこった方法が有効になる．バッチ処理システムなどで facility の数が限られている場合には，文献 7) の方法が有効であろうが，TSS などでは，文献 8) に準じた方法がよい．EMPS では，ETSS と同じく，ロックされた facility の状況を監視し，deadlock が生ずるロック要求は，適宜拒否するようにしている．

8. むすび

EMPS は，測定の結果，多重プロセッサシステムとして約 1.4 台注) の効率で動くことがわかった[9]．この数字は，測定がシステムにとってきびしい負荷のもとで行なわれたこと，ハードウェア自体，改造機のため多重プロセッシングとしてはあまり効率のよくないことを考慮すると，かなり高いものである．したがって，基本方針の正しさ，ここで提案した諸方式の有効性が，幾分なりとも実証されたものと信じている．なお，EMPS では，多重プロセッサシステムのもう一つの側面である信頼性の向上の処置は最小限にとどめた．その理由の一つは，信頼性向上のためには，相応のハードウェアの機能がなければ，ソフトウェアだけではさほど有効な処理はとれないからである．

昨今，処理装置において，演算および制御装置の値段が主記憶に比べ，ますます安くなる傾向にあることを考えると，EMPS の成果は，多重プロセッサシステムの有効性を証明したことになる．

注) メモリスイッチのためにサイクル時間から計算すると，0.24 台の効率低下がおこる．したがって 0.36 台分が，メモリ上での競合および interlock による効率低下の部分である．なお，この測定値は，多数の測定結果から2台の処理装置が 100% 並列に稼動した場合の値として推定した値である．

謝 辞

最後に，この研究の機会を与えてくださり，ご支援くださった，野田電子計算機部部長，並びに西野ソフトウェア部部長とご討議くださった研究室の諸氏に厚く感謝する．

参考文献

1) 淵：時分割共同利用システム—ETSS について，情報処理，11，4，p. 219，(1970-4)
2) Lampson, B. W.: A Scheduling Philosophy for Multiprocessing Systems. CACM, 11, No. 5. p. 347 (May 1968)
3) Witt, B. I.: M 65 MP: An Experiment in OS/360 Multiprocessing, Proc. of ACM National Conference, p. 691 (1968)
4) Smith, K. C.: A dual processor checkout system, Proc. FJCC, p. 1177 (1968)
5) Gibson, C. T.: Time-sharing in the IBM System/360: model 67, Proc. SJCC, p. 61 (1966)
6) Havender, J. W.: Avoiding deadlock in multitasking systems, IBM System Journal, No. 2, p. 74 (1968)
7) Habermann, A. N.: Prevention of System Deadlock, CACM, 12, No. 7, p. 373 (Jul. 1969)
8) Murphy, J. E.: Resource allocation with interlock detection in a multitask system, Proc. FJCC, p. 1169 (1968)
9) 横井，他：HITAC-8410 による多重プロセッサシステム (EMPS)，OS シンポジウム報告集（情報処理学会）(1970-9)