ショート・ノート

並列プロセスの制御問題とその応用*

斎　藤　信　男**

Abstract

A solution of the mutual exclusion problem in concurrent processes was first given by Dijkstra and was then modified by Knuth. This paper proposes another solution to this problem.

The author then discusses how to apply this solution to the control problem in a poly-processor system. The new solution gives a correct implementation algorithm of the indivisible operation if a poly-processor system has no shared memory and is organized as a ring structure via communicatcn lines.

1. はじめに

並列的に処理されるn個の逐次型プロセスの相互排除問題は，Dijkstra[1]により最初に解かれ，Knuth[2]によって見かけ上のデッドロックをなくすように修正された．見かけ上のデッドロックの除去を効率よく行う解が，Eisenbergら[3]によって提案されている．この論文では，Knuthの解の修正案を示し，ある結合構造を持った複合プロセサ・システムにおける非可分操作の実現に際し，その解が利用できることを論ずる．

ALGORITHM 1

```
global integer array control[1:n]; integer k;
comment behavior of process i;
begin integer j;
  L0: control[i]:=1;
  L1: for j:=k step -1 until 1, n step -1 until 1 do
        begin if j=i then go to L2;
            if control[j]≠0 then go to L1
        end;
  L2: control[i]:=2;
      for j:=n step -1 until 1 do
        if (j≠i) and (control[j]=2) then go to L0;
  L3: k:=i;
      critical section;
      k:=if i=1 then n else i-1;
  L4: control[i]:=0;
  L5: remainder of cycle in which stopping is allowed;
      go to L0
end
```

Fig. 1 algorithm 1 (by D.E. Knuth)

2. 新しいアルゴリズム

Knuthの提案した解(アルゴリズム1)を**Fig. 1**に示し，新しい解（アルゴリズム2）を**Fig. 2**に示す．アルゴリズム2がn個のプロセスの相互排除問題に関する正しい解であることを主張するためには，次の四つの条件が成立することを示せばよい．

（1） critical section (CS) に対する相互排除性が保証されている．(mutual exclusiveness)

ALGORITHM 2

```
global integer array control[1:n]; integer k;
comment behavior of process i;
begin integer j;
  L0: control[i]:=1;
  L1: for j:=i+1 step 1 until n, 1 step 1 until i-1 do
        begin if j=k then go to L2;
              if control[j]≠0 then go to L1
        end;
  L2: control[i]:=2;
      for j:=i+1 step 1 until n, 1 step 1 until i-1 do
        if control[j]=2 then go to L0;
  L3: k:=i;
      critical section;
      k:=if i=1 then n else i-1;
  L4: control[i]:=0;
  L5: remainder of cycle in which stopping is allowed;
      go to L0
end
```

Fig. 2 algorithm 2

---

* Control Problem of Concurrent Processes and Its Application by Nobuo SAITO (Institute of Electronics and Information Science, The University of Tsukuba)

** 筑波大学電子情報工学系

(2) 任意の順序で CS に入ることができる. (arbitrariness)

(3) 全体が停止してしまう (誰も CS に入れない) ことはない. (deadlock freeness)

(4) どれか一つのプロセスが停止してしまう (そのプロセスが, いつまでも CS に入れない) ことはない. (effective deadlock freeness)

なお, 各プロセスは, remainder の中で永久に停止してもよいことを仮定しておく.

(1)の条件: L3 の CS に到達したときは, (control[i]=2)∧{$\forall j(i \neq j)$(control[j]≠2)} が成り立っており, また control[i]=2 が成り立っていれば, どのプロセス $j(j \neq i)$ も L2 の for loop を抜けられないので, 二つ以上のプロセスが同時に L3 に到達することはあり得ない.

(2)の条件: プロセス i だけが動いていて, それ以外は remainder の内部で停っているものとする. このとき, control[j]($i \neq j$) の値は 0 であり, プロセス i は L1 および L2 の for loop を抜けることができる. これは, i の選択の如何に拘わらず成り立つので, CS に入るのは任意の順序で行われるものと考えてよい.

(3)の条件: この手続き内の四つの loop, すなわち, L1 の for loop (loop 1), L1 の for loop 内で L1 へ戻ることにより生ずる loop (loop 2), L2 の for loop (loop 3) 及び L2 の for loop 内で L0 へ戻ることにより生ずる loop (loop 4) に対して, これらの loop 及びそれらの組み合わせによってできる loop を次々とくり返して, どれも CS に入れないという状況が生じないことを示す.

loop 1 では, loop 2 を生ずる条件が成り立つ場合以外は, L2 へ到達する.

loop 2 では, k に対する代入が L3 でしか行われないので, 全てのプロセスが loop 2 をくり返しているときは k の値は変化しない. したがって, k の値に等しい番号のプロセスで $j=k$ が成立し, L2 へ抜ける.

loop 3 では, loop 4 を生ずる条件が成り立つ以外には, L3 へ到達する.

loop 4 では, L0→L2→L0 という loop を永久にくり返す可能性がある. いま, プロセス i とプロセス j 以外は, remainder にあって control の値が 0 であると仮定する. また, i の方が j よりも k の値に近いものとする. プロセス i が一旦 L0 を通過すると, CS に達しない限り control[i]≠0 であるから, プロセス j は, L1 の for loop において $j=k$ が成立して L2 へ飛び越す前に control[j]≠0 が成立して必ず L1 へ飛び越してしまい, control[i]=0 になるまで L2 へ到達することができない. すなわち, control[j]≠2 が成立する. プロセス i は, L2 の for loop で control[j]=2 が成立して L0 へ戻ることは起こり得ず, 必ずこの for loop を抜けて L3 へ到達できる. control の値が 0 でないプロセスが 3 個以上あっても, k の値に一番近い番号のプロセスが必ず L3 へ到達できる.

(4)の条件: プロセス i が remainder で停っていないで CS へ入ることを要求しているのにも拘わらず, loop 1～4 のどれかまたはその組み合わせで生ずる loop をくり返しているものとする. このとき, control[i]≠0 が常に成立している.

他のプロセスが L1 の for loop を抜け出すのは, $j=k$ が成立するときのみである. しかも, L1 の for loop を実行しているプロセスの中では, k に一番近い番号のプロセスが抜けてくる. 一方, CS から抜け出た後, k の値は 1 つ減少させられるので, CS へ入ることを要求しているプロセスの中では, i が k の値に一番近くなる状態がいつかは起こり, i は L1 の for loop を抜け出すことができる.

L2 に到達しても, for loop 内で control[j]=2 ($j \neq k$) が成り立つと L0 へ戻ってしまい, そのとき他のプロセスが CS の後で k の値を変更する可能性がある. しかし, k の値は 1 つしか減らされず, またそのとき CS に入っていたプロセスは, i よりも大きな番号を持っていたはずであるから, プロセス i が CS に到達する以前に k の値が i よりも小さくなることは決して生じない. したがって, L1 で再び for loop を実行すると, プロセス i が最初に抜けてくる.

3. 複合プロセサ・システムの制御への応用

複数個のプロセサを結合した複合プロセサ・システムは, 新しい計算機システムの構成法として最近注目をあびているが, そのようなシステムの OS の核は, 同期基本命令を実現するために共有変数と非可分操作とを実現する必要がある[4]. この二つの条件を実現する方法は, 複合プロセサの結合方式によって大きく左右される.

プロセサ間のハードウェア的な結合は, (1)通信線(または, 割り込み線)を介すること, (2)共有記憶を介することの二つの方法を適当に組み合わせて行われ

る．通信線を介する場合，一般に，割り込まれる側のプロセサは割り込み禁止の機能を備えている．また，通信線による結合の位相構造は，完全グラフ構造，リング構造，何らかの階層構造などが考えられる．一方，共有記憶を介する場合，それがテスト・アンド・セット (test and set——TS) 命令を持っているかどうかが重要な問題になる．

複合プロセサ・システムの結合方式を上述の機能に従って分類した場合，非可分操作を正しく実現する方法は **Table 1** の如くになる．アルゴリズム1においては，n 個の要素から成る配列 control [1 : n] と，一つの整数変数 k とを全域的変数として用いる．(3)—1 および(3)—2 の場合には，共有記憶上にこれらの変数を実現することができる．(3)—3 の場合，control [i] に対する代入操作はプロセサ i でしか行われないので，control [i] はプロセサ i の局所記憶の中で実現し，他のプロセサからの参照は通信線を介して行う．また，k に対する代入操作はプロセサ i の危険部分の内部でのみ行われるので，k に対するロックをする必要はなく，k を実現する共有記憶には TS 機能を設ける必要はない．

Table 1 の(4)の場合にアルゴリズム1を用いた場合，配列の各要素 control [i] をプロセサ i の局所記憶に実現すると，たとえば L1 の for loop において control [i] の値を判定するときに任意のプロセサ間の通信線が存在しないので困難になる．アルゴリズム2を用いれば，L1 の for loop において control [i] の値の判定のくり返しを次々と隣りのプロセサに送ってゆくことが可能となる．また，全域的な整数変数 k を用いなければならないが，これをどこかのプロセサの局所記憶に実現すると，それに対する代入操作や参照の要求をリング構造上のプロセサ間で送信しなければならず，それが正しく行われるためには再び非可分操作を実現しなければならない．この手順を何回くり返しても，正しい実現には到達できない．したがって，**Fig. 3** に示すように一つの変数 k を全プロセサに共通な共有記憶として実現することが不可欠の条件となる．変数 k に対する代入操作は各プロセサ i の危険部分の中でのみ行われるので，k に対するロックをする必要はなく，共有記憶に TS 機能を設ける必要はない．

Table 1 Implementation of Indivisible Operation

通信線による結合 ＼ 共有記憶		共有記憶あり		共有記憶なし
		TS 機能あり	TS 機能なし	
階層構造		(1)—1 TS 命令の利用	(2)—1 割り込み禁止機能の利用	(2)—2 割り込み禁止機能の利用
民主構造	完全グラフ構造	(1)—2 TS 命令の利用	(3)—1 アルゴリズム1	(3)—3 アルゴリズム1
民主構造	リング構造	(1)—3 TS 命令の利用	(3)—2 アルゴリズム1	(4) アルゴリズム2

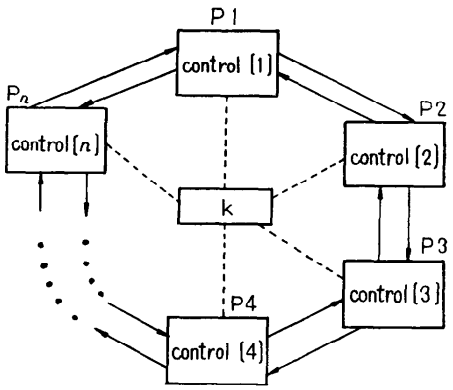

Fig. 3 ring structure

4. おわりに

n 個のプロセスの相互排除問題の新しい解を示し，複合プロセサ・システムにおける非可分操作の実現に対する応用性を論じた．アルゴリズム1またはアルゴリズム2を用いた実現はかなりのオーバーヘッドを生ずるであろうが，実際のシステムの構成に際しては，ハードウェアによる TS 命令の実現や共有記憶の実現のコストと比較し，全体として効率のよい設計をしなければならない．

参考文献

1) E. W. Dijkstra: Solution of a Problem in Concurrent Programming Control, CACM, Vol. 8, No. 9, p. 569 (1965).
2) D. E. Knuth: Additional Comments on a Problem in Concurrent Programming Control, CACM, Vol. 9, No. 5, pp. 321～322 (1966).
3) M. A. Eisenberg and M. R. McGuire: Further Comment on Dijkstra's Concurrent Programming Control Problem, CACM, Vol. 15, No. 11, p. 999 (1972).
4) 斎藤: 同期基本命令の実現について，情報処理，Vol. 15, No. 11, pp. 841～849 (1974).

(昭和 50 年 10 月 6 日受付)
(昭和 51 年 3 月 24 日再受付)