デッドロックの検出法について*

穂　鷹　良　介**

Abstract

A new method for detecting and characterizing system deadlocks is presented.

An ordinary matrix representation for a finite directed graph is used to detect the existence of deadlocks. I.e., among n concurrent jobs, deadlocks exist if and only if $A^n \neq 0$ (where A denotes the matrix used for representing the graph).

1. はじめに

multiprogramming を行なっているとき，ジョブ1が直接的あるいは間接的にジョブ2に割り当てられている資源（これをBとしよう）を要求し，逆にジョブ2が直接的あるいは間接的にジョブ1に割り当てられている資源（これをAとしよう）を要求したとしよう．またさらに，どちらのジョブも資源A, Bをそれぞれ独占して使用しなくてはならないものとする．このときにはどちらのジョブも自分の要求する資源を占有しているジョブが処理を終了してその資源を開放してくれるのを待つと三すくみのような状態になって，いつまで待ってもラチがあかない，いわゆるデッドロックの状態となる．

デッドロック状態については，その予防法・解決法など数多くの方法（参考文献を参照のこと）があるが，筆者は，最近オンライン・データ・ベースの開発に伴ってこの問題に直面し，自分なりにデッドロック状態の特徴づけをすることができたのでその成果について発表する．

3. で使うために若干の記号の用意をしておく．

A を $n\times n$ の行列とするとき

$$A^k=\underbrace{A\times A\times\cdots A}_{k\text{個}}=\begin{pmatrix} a_{11}^k & a_{12}^k & \cdots & a_{1n}^k \\ \vdots & \vdots & & \vdots \\ a_{n1}^k & a_{n2}^k & \cdots & a_{nn}^k \end{pmatrix}$$ (注)

また記号0で単に実数としての0のほかに，適宜ベクトル，行列などの0元も表わすものとする．

デッドロック状態については，たとえば参考文献に見られるようにいろいろの立場から研究がなされている．文献 1) では独立に実行される各タスクごとに使用する資源を表の形に整え，資源の競合関係から生ずるタスク間の待合せのすべてのケースを調べ，ある一つのタスクが直接または間接に自分自身を待ち合わせるときにデッドロックがあるものとして検出する方法を述べている．

文献 2), 3), 4) ではデッドロックの検出というよりは，デッドロックを起こさないための十分条件の提示がなされている．文献 5) は 4) の方法に対しての改善策の提案である．本論文で述べるものは文献 1) で取り扱われている問題に近いもので，仮定されているシステムでデッドロックが生ずるかどうかを厳密に判定する手段である．

2. 考察の対象

いま，問題とするシステムには同時にn個のジョブ$J_1, J_2, \cdots, J_n$が動作しているものとする．またこれらのジョブによって使用される資源を$R_1, R_2, \cdots, R_m$とする．

資源の使用モードには普通，排他的モード，共用的モードとがある．排他的モードの場合には1つのジョブがある資源Rを使用しているときには，どんな目的であろうが，ほかのジョブにRの使用を認めない．共用的モードの場合には，同じモードで使用する場合に限り，ほかのジョブが同一資源を使用することを認める．

これらの資源使用状態を制御するためには，われわれのやり方では，各資源ごとに資源の管理表を設け，そこにジョブからの資源要求が出されるたびにジョブ番号と資源の使用モードとを登録し，資源開放のメッ

---

* On a method of detecting a deadlock by Ryosuke Hotaka (Nippon Software Co. Ltd.)

** 日本ソフトウエア株式会社

(注)　したがって $a_{ij}^k=\sum_{i_1=1}^{n}\sum_{i_2=1}^{n}\cdots\sum_{i_{k-1}=1}^{n}a_{ii_1}a_{i_1i_2}\cdots a_{i_{k-1}j}$

セージが出されるたびに上記の登録を抹消する．どのジョブも資源 R を使用していない状態のときに，ジョブ J がその資源 R を使用したいという要求を出したときにはその要求が直ちに認められ，資源 R の使用モードはジョブ J の要求した使用モード M と等しくなる．

この状態でさらにジョブ J' が R を使用モード M' で使用する要求を出したときには次のようになる．

$M=M'$ でしかも M が共用モードである場合には J' の要求も直ちに認められ，R の資源管理表に J' が使用モード M' で使用中という状態が表示される．

それ以外の場合には M もしくは M' のいずれかが排他的モードであるから資源の同時使用は認められず，単純に考えるならば，J' の要求は，J の処理が終了するまで待たされることになる．デッドロックになるかどうかは，途中で上のような資源と，その使用モードについての問題はあるにせよ結局上のような形で，ジョブ J' が J を待つか待たないかという関係が問題であり，介在する資源に対しての考慮はデッドロックを検出するアルゴリズムの表面には出さないで済む．

そこで，一般にジョブ J_i がジョブ J_k を待つという関係を

$$J_i > J_k$$

と表わすことにする．

いま仮に **3.** で述べるような方法によって J' の要求を認めてもデッドロックにならないものと判断できたとして，J' の要求が R の資源管理表に入れられたとしよう．

さらに J'' が排他的なモードで資源 R の使用を要求してきたとき，優先順位がとくに定めてなく，FIFO (first-in first-out) で処理が行なわれるものとすると，すでに存在している関係

$$J' > J$$

のほかにさらに

$$J'' > J'$$

がシステムとして許されるかどうかというようなことが，われわれの問題の対象である．

3. 問題の数式化

$n \times n$ の行列 A を考える．A の元 a_{ij} は，$J_j > J_i$ と定めたとき 1，それ以外のときは 0 の値を持つものとする．

この場合，簡単のため $J_j > J_i$ というのは **2.** で述べた資源管理表の順序で，J_j が直前のジョブとしてジョブ J_i を待つときだけに上の行列で $a_{ij}=1$ としておくものとする．資源管理表の前の方にも J_i がさらに待っているジョブが多数あろうとも，それを行列 A の第 j 列の要素として 1 とはしないことにしておく．

いうまでもないが，いま J_{i_1} と J_{i_2} とが共用モードで資源 R を使用しているとき，J_j が排他的モードで R を使用するため待つとすると，

$$J_j > J_{i_1}$$

であり同時に，

$$J_j > J_{i_2}$$

であるから行列 A の第 J 列には 2 個の 1，$a_{i_1 j}$，$a_{i_2 j}$ が記入されることとなる．

さてこのようにして A が定義されたあとは，このジョブ間にデッドロックがあるかどうかということは，次の条件が成立するかどうかということにほかならない．

定義 行列 A を持つジョブ間にデッドロックが存在するとはある整数列 $\{i_\nu\}_{\nu=0,\cdots,n}$ $(1 \leq i_\nu \leq n)$ と整数 k $(1 \leq k \leq n)$ が存在して

$$a_{i_0 i_1} \cdot a_{i_1 i_2} \cdot \cdots \cdot a_{i_{k-1} i_k} = 1, \quad i_0 = i_k$$

が成立することである．

これは，

$$J_{i_k} > J_{i_{k-1}} > \cdots > J_{i_1} > J_{i_0}$$

であることと同値であるからジョブ J_{i_0} が自分で自分を待つことになりデッドロックとなっている．

ついでであるが，あるジョブ間にデッドロックがあれば，必ずジョブ $J_{i_1}, J_{i_2}, \cdots, J_{i_{k-1}}$ もひきずられてデッドロックになっていることに注意しておこう．

定理 一般に $n \times n$ の非負正方行列 A において，次の（A）または（B）が成立する．

（1） $A^n = 0$

（2） ある整数 k, i が存在して $(1 \leq i,\ k \leq n)$

$$a_{ii}^{k} > 0$$

証明 （1）を否定するとある整数 i_0, i_n $(1 \leq i_0, i_n \leq n)$ が存在して

$$a_{i_0 i_n}^{n} \neq 0$$

$$a_{i_0 i_n}^{n} = \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \cdots \sum_{i_{n-1}=1}^{n} a_{i_0 i_1} \cdot a_{i_1 i_2} \cdot \cdots \cdot a_{i_{n-1} i_n}$$

であるから，ある整数列 $\{i_\nu\}_{\nu=1,\cdots,n-1}$ $(1 \leq i_\nu \leq n)$ が存在して

$$a_{i_0 i_1} \cdot a_{i_1 i_2} \cdot \cdots \cdot a_{i_{n-1} i_n} \neq 0$$

となる．

数列 $\{i_0, i_1, \cdots, i_n\}$ の要素はすべて有限集合 $\{1, 2, \cdots, n\}$ から取り出してきた数値であるから，ある整数 p, q があって

$$1 \leq p < q \leq n$$
$$i_p = i_q$$

となる．

いま $k=q-p$, $i=i_p=i_q$ とおくと

$$a_{ii}^{k} = \sum_{j_1=1}^{n} \sum_{j_2=1}^{n} \cdots \sum_{j_{k-1}=1}^{n} a_{ij_1} \cdot a_{j_1 j_2} \cdot \cdots \cdot a_{j_{k-1} i}$$
$$\geqq a_{i_p i_{p+1}} \cdot a_{i_{p+1} i_{p+2}} \cdot \cdots \cdot a_{i_{q-1} i_q} \neq 0$$

A は非負行列であるから $a_{ii}^{k} > 0$ となる．Q.E.D.

この定理から次のことがわかる．ジョブ間の待ちを表現した行列は0と1の要素から成るから，明らかに上の定理の性質を満足しているが，さらに

$$a_{i_0 i_1} \cdot a_{i_1 i_2} \cdot \cdots \cdot a_{i_{k-1} i_k} = 1$$

と

$$a_{i_0 i_1} \cdot a_{i_1 i_2} \cdot \cdots \cdot a_{i_{k-1} i_k} > 0$$

とが同値となる．

したがって，この場合ジョブ間にデッドロックがあるかないかは行列 A を n 乗して全要素が0でないかどうかを調べることによって判別がつくことになる．

なお，デッドロックにひきこまれるジョブが k 個集まって $J_{i_1} > J_{i_2} > \cdots > J_{i_k} > J_{i_1}$ という関係をみたすときには，この行列の乗算を k 回行なったところで，ジョブ $J_{i_1}, J_{i_2}, \cdots, J_{i_k}$ に対応する積行列の主対角線上の要素は正となっていることに注意しておこう．

4. implementation

このデッドロックの検出方法は，一度ジョブ間の待ちを表現した行列 A が決定すればあとはいかに高速に行列の乗算を実行するかにある．これには通常の乗算をやる必要はなく，内部2進の計算機ではビットごとの AND をとり，結果が0かどうかを判別するだけでこと足りる．最後の決定は，A^n が0かどうかで判別されるかで，その行列要素の植の大きさは問題にならないからである．

実際の演算では最初に A の転置行列 A^T とを作っておいて，次々に作られる A のべき乗 A^k と A^T との行ごとの AND をとって0かどうかの判定をするのがよいと思われる．

システム全体を起動させるにはすべてのジョブに資源の使用を全然認めない状態から始めて，1つ1つジョブの資源の使用要求を上記の手続きを踏みながら認めて行く．このようにすれば，仮にデッドロック状態が検出されたときには，最後に新たに資源の要求を出したジョブがデッドロックをひき起こしたものと考えてよいから，しかるべく処置をすればよい．

5. おわりに

この問題に対して活発な議論をして筆者を助けて下さった日本ソフトウエア，小林昌之，鳩宿 憲，森田勝弘の諸氏に感謝します．

参考文献

1) James E. Murphy, "Resource allocation with interlock detection in a multi-task system" FJCC '68 1169-1176.
2) J. W. Havender, "Avoiding deadlock in multi-tasking systems" IBM SYST. J. No. 2 '68 74-84.
3) A. Shoshani and A. J. Bernstein, "Synchronization in a parallel-accessed data base", CACM Vol. 12 No. 11 '69 604-607.
4) A. N. Habermann, "Prevention of system deadlocks", CACM Vol. 12 No. 7 '69 373-385.
5) Richard C. Holt, "Comments on prevention of system deadlocks" CACM Vol. 14 No. 1 '71 36-38.

(昭和46年5月17日受付)