On a Method of Detecting a Deadlock

RYOSUKE HOTAKA*

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 $A=0$ (where A denotes the matrix used for representing the graph).

1. *Introduction*

Suppose n jobs $J_1, J_2, \cdots, J_n$ are concurrently operating in a multiprogramming environment.

If J_1 makes a dynamic request of using a resource R and if J_2 happens to be using that resource, there occur two cases.

case A. Resource R can be used by both J_1 and J_2.

case B. Otherwise.

In the case A, there is no problem. In the case B, if the supervisor admits job J_1 to wait job J_2, there occurs the danger of deadlocks.

For, suppose J_2 dynamically require another resource R'. In this case, if R' is exclusively used by J_1, J_2 waits J_1 and J_1 waits J_2 endlessly.

Though the deadlock occurrs concerning the use of resources, the distinction whether the deadlock is occurring in the system or not can be detected by observing the relation between jobs.

We express the relation "the job J_i waits job J_k" by

$$J_i > J_k.$$

We shall define the deadlock between $J_1, \cdots, J_n$:

Definition: A deadlock exists between $J_1, \cdots, J_n$ if there is a k and an integer sequence $\{i_\nu\}$ $\nu=0, 1, \cdots, n$ $(1 \leqq i_\nu \leqq n)$ such that the relations

$$J_{i_k} > J_{i_{k-1}} > \cdots > J_{i_1} > J_{i_0} \quad \text{and} \quad i_0 = i_k$$

hold.

2. *Algorithm*

We now present an algorithm which detects if there is a deadlock between jobs $J_1, \cdots, J_n$.

This paper first appeared in Japanese in Joho-Shori (Journal of the Information Processing Society of Japan), Vol. 12, No. 10 (1971), pp. 602~604.

* Nippon Software Co. Ltd.

Let us define $n\times n$ matrix $A=(a_{ij})$ as follows.

$$a_{ij}\begin{cases}1 & \text{if } J_i>J_j\\ 0 & \text{otherwise.}\end{cases}$$

It is easy to see that a deadlock exists between $J_1, \cdots, J_n$ if and only if there is an integer k and an integer sequence $\{i_\nu\}_{\nu=0\cdots n}$ $(1\leqq i\leqq n)$ such that

$$i_0=i_k$$

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

Theorem. Let A be nonnegative $n\times n$ matrix. Then we have

(1) $A^n=A\cdot\cdots\cdot A=0$

or

(2) There exist integers i, j and k $(1\leqq i,\ j,\ k\leqq n)$ such that $a^k{}_{ij}>0$, where $a^k{}_{ij}$ denotes the ij-th component of the k-th power of A.

Proof. Suppose (1) is not true. Then there are integers i_0, i_n $(1\leqq i_0,\ i_n\leqq n)$ such that

$$a^n{}_{i_0,i_n}\neq 0$$

$$a^n{}_{i_0,i_n}=\sum_{i_1=1}^{n}\sum_{i_2=1}^{n}\cdots\sum_{i_n=1}^{n}a_{i_0i_1}\cdot a_{i_1i_2}\cdot\cdots\cdot a_{i_{n-1}i_n}$$

Hence there is an integer sequence $\{i_\nu\}$ such that

$$1\leqq i_\nu\leqq n\ (\nu-1, \cdots, n-1)$$

$$a_{i_0i_1}\cdot a_{i_1i_2}\cdot\cdots\cdot a_{i_{n-1}i_n}\neq 0.$$

From this it is easy to see that there are integers p, q such that

$$0\leqq p<q\leqq n$$

$$i_p=i_q$$

Let $k=q-p,\quad i=i_p=i_q.$

Then

$$a^k{}_{ii}=\sum_{j_1=1}^{n}\sum_{j_2=1}^{n}\cdots\sum_{j_{k-1}=1}^{n}a_{ij_1}\cdot a_{j_1j_2}\cdot\cdots\cdot a_{j_{k-1}i}\geqq a_{i_pi_{p+1}}\cdot\cdots\cdot a_{i_{q-1}i_q}\neq 0$$

As A is nonnegative $a^k{}_{ii}>0$. Q.E.D.

In the matrix A which represents the waiting conditions between jobs,

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

and

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

are equivalent.

Hence we can tell if there is a deadlock between $J_1, \cdots, J_n$ by multiplying A n times.

3. *Implementation*

The calculation of A^n can be done effectively as follows. It is not necessary to execute ordinary multiplication. It is enough to take AND operation bitwise and detects if the result is 0.

From the theorem, we can detect the existence of a deadlock by investigating if $A^n=0$ or not. The value of A^n is not our problem.

In reality, we should make A^T at first. Bitwise AND operation between the row of A^k and A^T will bring the result.

4. *Acknowledgement*

The author is grateful to Messrs M. Kobayashi, K. Hatoya and K. Morita for their useful comments.