System deadlocks resolution

by KOJI NEZU
Nippon Electric Co., Ltd.
Kawasaki, Japan

INTRODUCTION

With rapid large online systems growth, the system deadlocks problem is becoming of major importance. Sharing on increasingly sophisticated set of services and resources among an increasing number of terminal users leads to an increase in the chance for deadlock occurrence.

Two approaches for the problem are: (1) deadlock occurrence prevention or avoidance; (2) early deadlock detection and quick recovery.

Approach 1 is ideal, if it would be completely realized within an allowable system performance degradation limit. However, a system design to fulfill this requirement is very difficult in most online systems. Typical is multiaccess database. For this reason, approach 2 must be studied and a practical method that realizes fully automatic deadlock detection and recovery must be developed.

A general deadlock detection algorithm has been known. Moreover, high speed hardware which simulates the algorithm has been proposed. However, no general algorithm has been developed for recovery.

Recovery from system deadlocks is carried out by aborting appropriate deadlocked processes, freeing the resources that had been exclusively allocated to these processes and making it possible to run the other deadlocked processes whose execution has been blocked by a round robin wait for resources among deadlocked processes.

As an example, assume two processes, Pa and Pb, are detected to be mutually deadlocked. It is obvious that there are two solutions for resolving this problem, one is process Pa abortion, and another is process Pb abortion. When there are more than two deadlocked processes, the existence of multiple solutions may also be expected. In fact, as proven in the following section, multiple solutions always exist.

The above discussion indicates that the following three steps are necessary for recovery: (step 1) obtain solutions; (step 2) select one solution to actually aborted processes and execute abort; (step 3) restart aborted processes at appropriate instants.

In order to accomplish an optimum decision at step 2, all the solutions should be known for step 1.

This paper presents a new systematic method for obtaining all the solutions for step 1.

Further discussion about steps 2 and 3 does not appear in this paper. The papers indicated in the references will be helpful for understanding these subjects.

PRELIMINARIES

To clarify the discussion, consider the following system model, which was proposed by Habermann.

A set of sequential processes which share system resources in such a way that, while resource A is allocated to process P1, no other process Pj can seize A, and an allocated resource is not released until it has fulfilled its task.

Generally every resource is not necessarily unique. Therefore, consider such a system that consists of n processes P1, P2,...,Pn and m different kinds of resources (n~1,m~1).

Although deadlock detection is not the subject of this paper, a brief description about the detection algorithm is also necessary.

The deadlock detection algorithm given by Shoshani and Coffman is summarized as follows. The first step is to pick up processes which are requesting no specific resources or requesting only free resources. It is clear that these processes are not deadlocked. The second step is to assume the resources allocated to the picked up processes as "free resources," and go back to the first step. By this loop, all the non-deadlocked processes are picked up. Therefore, if all the processes are confirmed to be non-deadlocked, no deadlock exists in the system. Otherwise, deadlock exists and remaining processes are deadlocked.

SOLUTIONS

It can be said that the deadlocked state complexity level is closely related to the number of deadlocked processes. When there are two deadlocked processes, all solutions are immediately known. As illustrated in the preceding section, that is Pa and Pb are deadlocked mutually, solutions are {Pa} and {Pb}. However, when there are more than two deadlocked processes, probably all the solutions cannot be obtained immediately. Up to now, the term 'solution' has been used in a tacit sense. Now, let us define it clearly as follows:

"A solution is the smallest subset of deadlocked processes, whose aborting resolves the deadlock."
For example, consider that the deadlock in which four processes $P_1, P_2, P_3,$ and $P_4$ fall, and that, by some means, it is confirmed that, by aborting both $P_1$ and $P_2$, deadlock is resolved, but that by aborting only $P_1$ or $P_2$, the deadlock is not resolved. Then, $\{P_1, P_2\}$ is a solution. Obviously by aborting $\{P_1, P_2, P_3\}$ it is resolved, but aborting of $P_3$ in addition to $\{P_1, P_2\}$ is meaningless. Above definition eliminates $\{P_1, P_2, P_3\}$ from solutions.

Consider in the same example, $\{P_2, P_3, P_4\}$ is aborted. Obviously, the deadlock is resolved because the only remaining process is $P_1$. As $P_1$ is an element of above solution, this means another solution, which does not involve $P_1$, exists. From this example, the following general theory is induced.

“When deadlock occurs, multiple solutions always exist.”

**OBTAINING A SOLUTION**

Among the multiple solutions, one solution can be obtained by the following procedure;

**Procedure 1**

Step 1: Arrange all the deadlocked processes linearly in an arbitrary order, such as $(P_d_1, P_d_2, \ldots, P_d_n)$.

Step 2: Scan from $P_d_1$ to $P_d_n$, marking such processes that must be aborted for deadlock resolution, if no un-marked processes, those that have been scanned, are aborted.

Step 3: Collect all the marked processes. This set is one of the solutions.

It may be easily understood that, by aborting the above obtained set of processes, deadlock is resolved, but that by aborting any true subset, it is not resolved. Therefore, the set of above marked processes is a solution.

Note that a solution can be obtained in a straightforward way.

**Example 1**

Obtain a solution for the system deadlock shown in Table I.

<table>
<thead>
<tr>
<th>RESOURCE NAME</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
<th>$R_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER OF RESOURCES</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>RESOURCES</td>
<td>$P_1$</td>
<td>1/0</td>
<td>1/0</td>
<td>0/3</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0/2</td>
<td>1/0</td>
<td>1/0</td>
<td>0/1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>0/0</td>
<td>1/0</td>
<td>0/2</td>
<td>0/0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>2/0</td>
</tr>
<tr>
<td>$P_5$</td>
<td>1/0</td>
<td>0/3</td>
<td>2/0</td>
<td>0/1</td>
</tr>
</tbody>
</table>

* requested but not allocated to the process

By applying Shoshani and Coffman’s algorithm explained in the previous section, it is known that, except for $P_2$, which is requesting no resource, all other processes are mutually deadlocked.

Using procedure 1, a solution for the deadlock resolution can be obtained.

Step 1: $(P_1, P_2, P_3, P_4)$

Step 2: $P_1$: not marked. *The deadlock is resolved by aborting $P_2, P_3, P_5$.*

$P_2$: marked. *Even if $P_3$ and $P_4$ are aborted, deadlock is not resolved. (Use Shoshani and Coffman’s algorithm hereafter.)*

$P_3$: not marked. *The deadlock is resolved by aborting $P_2$ and $P_5$.*

$P_4$: marked. *The deadlock is not resolved by aborting $P_2$.*

Step 3: one solution = $\{P_1, P_2\}$

Note that the solution depends on the process order arranged in step 1. Therefore, denote each solution, suffixing the process order, such as $\alpha(1, 2, 3, 5) = \{P_1, P_3\}$.

**ALL SOLUTIONS**

Another solution for the above example can be obtained by placing one of the processes contained in the above solution $\{P_2, P_3\}$ to the left end of the line in step 1, such as $(P_2, P_1, P_3, P_4)$, and then executing step 2. Obviously, the new solution, that is $\alpha(2, 1, 3, 5)$, is different from the previous solution, because $\alpha(2, 1, 3, 5)$ does not contain $P_2$. For the third and further solutions, the same basic idea can be applied. However, existence of more than two solutions is not guaranteed. Therefore, before applying procedure 1, confirming step is necessary.

The following theorem is useful for this objective.

“If there is any set of deadlock processes which, for every already known solution, contains at least one same process, where the deadlock can be resolved without aborting this set, at least one new solution exists. Otherwise, no new solution exists.”

The first half may be trivial. The latter half is proven as follows: From the definition of “solution,” a solution is neither subset nor superset of another solution. Therefore, a new solution, when it is compared with an already known solution, does not contain at least one deadlocked process that is contained in the already known solution. This relation is maintained over all already known solutions.

**Example 2**

Obtain all the solutions for the system state shown in example 1.

The first solution has been obtained.

$\alpha(1, 2, 3, 5) = \{P_1, P_3\}$
The second solution is obtained by placing \( P_2 \) on the left end of the line in step 1 of procedure 1.

\[ \alpha_{o,1,3,5} = \{ P_1, P_5 \} \]

For the third solution, its existence must be checked. To apply the theorem, a check table is convenient.

Note that in Table II, all the combinations, which involve a single element picked up from each already known solution, are checked. The reason will be clear later.

From the table, it is confirmed that a new solution exists and does not contain \( P_5 \).

Placing \( P_5 \) at left end of the line, a new solution is obtained by procedure 1.

\[ \alpha_{o,1,2,3} = \{ P_1, P_2, P_3 \} \]

Existence of the fourth solution is checked in the same way. However, a direct check about all the combinations for the three already obtained solutions is not necessary, because all the combinations up to the second solution have been checked in Table II. An additional check must be made for the ‘YES’ row in Table II, that is \( P_3 \) row, and the third solution.

As all rows are ‘NO’, no further solution exists. Summing up, all the solutions are

\[ \{ P_2, P_3 \}, \{ P_1, P_3 \}, \{ P_1, P_2, P_3 \} \]

CHECK TABLE NORMALIZATION

In this section, redundant rows elimination from check tables is discussed.

Clearly, the objectives for making check tables is to pick up all the subsets of the deadlocked processes which contain new solutions. From this objective, it may be understood that the second and the third rows in Table II are redundant, because, in these rows, the subset for the fourth row are checked. More concretely speaking, at the second and third rows, solution existence in \( \{ P_1, P_3 \} \) or \( \{ P_2, P_3 \} \) respectively, is checked. At the fourth row, the solution existence in \( \{ P_1, P_2, P_3 \} \) is checked. If the fourth row is ‘YES,’ it is not necessary to check the second and the third row due to the above mentioned objectives. If it is ‘NO,’ the latter two rows are obviously ‘NO’ too. Therefore, no check is necessary at second and third rows.

In summary, check table simplification is accomplished by eliminating all rows which are supersets of another row in the same table.

This simplification is called “check table normalization.”

By applying “normalization,” the procedure shown in the example 2 can be executed more clearly and efficiently.

Normalization effectiveness becomes conspicuous for more complicated deadlocks.

Example 3

Assume, in example 1, that a new request for three \( R_2 \) resources is issued from \( P_4 \), and \( P_4 \) also falls into deadlock, as a result.

The first solution is

\[ \alpha_{o,1,2,3,4,5} = \{ P_1, P_4, P_5 \} \]

The second solution is

\[ \alpha_{o,1,2,3,4,5} = \{ P_1, P_2, P_3 \} \]

Check table for the third solution is:

\[ \begin{array}{c|c}
\text{DLP not being aborted} & \text{Can deadlock be resolved?} \\
\hline
P_1 & P_2 & NO \\
P_2 & P_4 & NO \\
P_2 & P_5 & NO \\
P_2 & P_4 & YES \\
\end{array} 
\]

Third solution:

\[ \alpha_{o,1,2,3,4,5} = \{ P_1, P_2, P_3 \} \]
Check table for the fourth solution is:

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_4$</th>
<th>$P_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_2$</td>
<td></td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_4$</td>
<td></td>
<td></td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>$P_5$</td>
<td></td>
<td></td>
<td></td>
<td>YES</td>
</tr>
</tbody>
</table>

Fourth solution: $\alpha_{1,2,3,4} = \{P_1, P_2, P_3\}$

Check table for the fifth solution is:

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_2$</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_3$</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No further solution exists.

CONCLUSION

To accomplish non-stop operation of large online systems, a new general algorithm, for obtaining deadlock resolution solutions, has been proposed. Experimental implementation on a multiaccess database system is planned at Central Research Laboratories, NEC.

ACKNOWLEDGMENT

The author would like to thank H. Kubo and M. Mizuma, both of Central Research Laboratories NEC, for their critical advice in preparation of this paper.

REFERENCES