Deadlock Prevention in Process Control Computer Systems

S. TSUTSUI* AND Y. FUJIMOTO**

* *Department of Management and Information Science, Faculty of Commerce, Hannan University, 5-4-33 Amamihigashi, Matsubara-shi, Osaka 580 Japan.*

** *Central Research Laboratories, Engineering Center, Sharp Corporation, 2613-1 Ichinomoto, Tenri-shi, Nara 632 Japan.*

System deadlock is a serious problem in a multiprogramming environment. The approaches to this problem can be divided into three categories: (1) prevention, (2) detection and recovery, and (3) avoidance. This paper proposes a variation of the first approach, partially applying ideas developed in the second and third approaches. This approach is especially effective in process control computer systems in which the application programs are usually fixed once designed. Using four predetermined application program parameters obtained in the program development stage, a directed graph model and a 'restriction' matrix model are introduced representing the usage of common resources. Conditions sufficient for system deadlock prevention are presented along with algorithms for checking to see that the models meet these conditions. By using this approach, if a deadlock possibility is detected the causes can also be detected. The deadlock can thus be prevented during the program development stage. As the algorithms are not used in the real-time mode, there is no negative effect on the responsiveness of the system. A higher utilisation rate of common resources is also ensured because the usage of resources is restricted only when the possibility of a deadlock is detected.

Received June 1985

1. INTRODUCTION

System deadlock is a serious problem in a multiprogramming environment in which a number of tasks simultaneously share more than one common resource. Deadlock situations arise when a group of tasks interlock with each other because of conflicting resource requirements. Thus, if deadlock situations arise in an online computer system, the system cannot respond within an acceptable period of time.

This is particularly true in process control applications, where a very quick response is required of computer systems. In these applications, a system deadlock may cause tremendous damage to the controlled plants.

System deadlocks have already been studied from various viewpoints. According to Coffman,[1] approaches to this problem can be classified into three categories: (1) prevention, (2) detection and recovery, and (3) avoidance.

In the prevention approach the usage of resources is restricted so that system deadlock will never occur.[2] This approach, however, has the disadvantage of degrading system performance, because of severe constraints on resource usage.

In the detection and recovery approach, all resource requests are granted. A control program is periodically executed to examine current resource allocations and to determine if there are any tasks in a deadlock state. If a deadlock is detected, a control program performs corrective measures.[3–6]

In the avoidance approach, a control program examines resource usage and grants the request only if the completion of all tasks can be guaranteed.[7–11]

There are also some problems in the last two approaches. One of the main problems is that the time overhead of the operating systems using these approaches tends to be greater because the control program must be executed in the real-time mode.

This paper proposes a variation of the first approach, partially applying the ideas developed in the second and third approaches. The proposed approach is especially effective in process control computer systems, in which the application programs are usually fixed.

Using four predetermined application program parameters obtained in the program development stage, a directed graph model and a 'restriction' matrix model are introduced representing the usage of common resources. Conditions for system deadlock prevention are presented with the algorithms for checking whether or not the models meet these conditions.

In this approach, if a deadlock possibility is detected the causes can also be detected. The deadlock can thus be prevented during the program development stage. As the algorithms are not executed in the real-time mode, there will be no negative effect on system responsiveness.

2. ASSUMPTIONS AND DEFINITIONS

2.1 Assumptions

Four predetermined parameters relating to the application programs must be given in the development stage.

(1) *Mode of usage for resources in each task.* Information must be given as to whether each task requests common resources in shared usage or exclusive usage. Definitions of shared usage and exclusive usage will be given in section 2.2.

(2) *Sequence of common resource usage in each task.*

(3) *Non-concurrent tasks.* In a multiprogramming environment, a number of tasks are activated and run simultaneously. However, there are pairs of tasks which are never activated concurrently with one other; for example, when a set of tasks is designed to be sequentially activated. As much information as possible relating to the non-concurrency between tasks is assumed to be available.

(4) *Types and number of common resources.*

In process control computer systems it is not difficult to obtain these parameters in the development stage of application programs.

2.2 Definitions

Definition 1. Exclusive usage and shared usage

Exclusive and shared usage are mutually defined as follows.

(1) *Exclusive usage.* A resource is said to be in an exclusive usage state when being used for only one task; the requests for this resource from other tasks will be suspended until the resource is released, whether the use requests are for exclusive or shared usage.

(2) *Shared usage.* A resource is said to be in a shared usage state when being used in more than one task simultaneously; the resource cannot accept an exclusive usage until all tasks performing a shared usage release control on it.

Definition 2. System deadlock

When a task requests a resource, the system is said to be in a deadlock state if the following two conditions, (1) and (2), are true at the same time.

(1) Either one of the two following situations is encountered. (*a*) A task requests exclusive usage of a resource, but the resource is already being used for exclusive or shared usage by one or more other tasks. The request is then suspended. (*b*) A task requests shared usage of a resource, but the resource is already being used for exclusive usage by another task. The request is then suspended.

(2) The request for the resource cannot be accepted unless the requesting task releases at least one of the resources which the task is holding.

Definition 3. The 'wait' relation between resources

If task T requests resource R_j while holding resource R_i ($i \neq j$), it is said that 'resource R_i is waiting for resource R_j in task T'. This situation is denoted by $R_i \xrightarrow{T} R_j$. This relation is referred to as the 'wait' relation between resources.

Definition 4. Propagation of wait relations

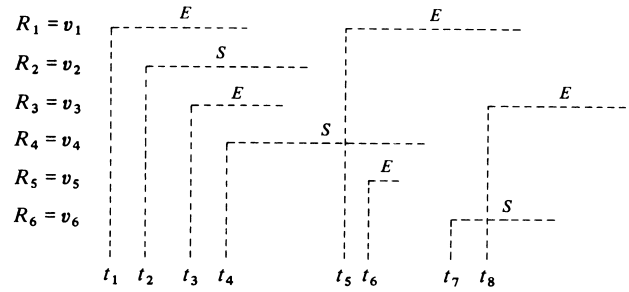
Let T_1 and T_2 be two tasks. If the following two wait relations, $R_i \xrightarrow{T_1} R_j$ and $R_j \xrightarrow{T_2} R_k$ are true, and the usage of resource R_j by T_1 and/or T_2 is in an exclusive usage state, it is said that 'resource R_i is waiting for resource R_k via resource R_j'. This situation is denoted by $R_i \xrightarrow{T_1} R_j \xrightarrow{T_2} R_k$. This relation is referred to as 'propagation of wait relation'.

3. DIRECTED GRAPH FOR WAIT RELATIONS AND RESTRICTION MATRIX

3.1 Directed graph for wait relations

Let $T_i (i = 1, 2, \ldots, N)$ represent the ith task and $R_j (j = 1, 2, \ldots, M)$ the jth common resource. After developing application programs, a Gantt Chart[12] is drawn representing the sequence and mode of resource usage for resources in each task, as shown in Fig. 1.

Next, a directed graph representing the wait relations described in Definition 3 is constructed from the Gantt Chart in the following manner.



E = exclusive usage, S = shared usage,
$t_1, t_2, \ldots, t_8$ = time point for resource requests.

Figure 1. Gantt Chart representation of use order and usage of resources in a task.

(1) Let $t_1, t_2, \ldots, t_r$ indicate the sequence of the time points at which task T_i requests the resources ($t_1, t_2, \ldots, t_8$ in Fig. 1). A directed subgraph $G_{ij} = [V_{ij}, A_{ij}]$ is obtained for each time point, whose vertices $V_{ij} = \{v_k\}$ correspond to the resources being held or requested at time point t_j by task T_i, and whose arcs $A_{ij} = \{a_l\}$ correspond to the wait relations between the resources at time point t_j. According to the mode of usage of resources v_p being held at t_j, and resource v_q requested at t_j, the arc $a_l(v_p \xrightarrow{a_l} v_q)$ is called an EE arc if v_p and v_q are E and E, ES if they are E and S, SE if they are S and E, and SS if they are both S (Fig. 2).

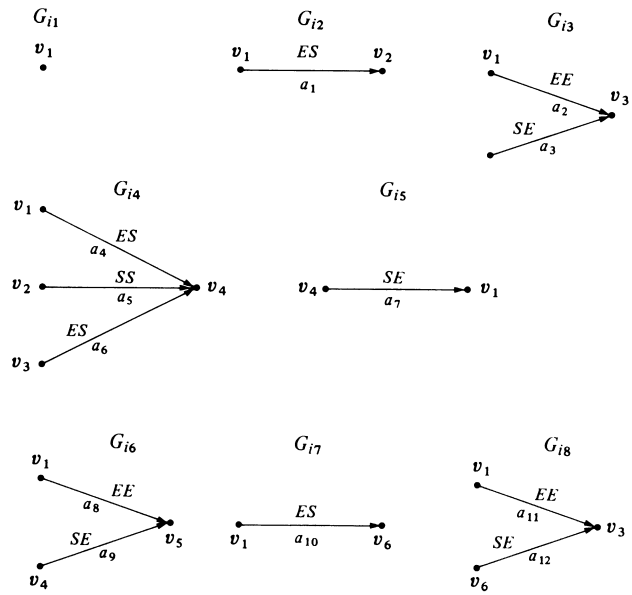
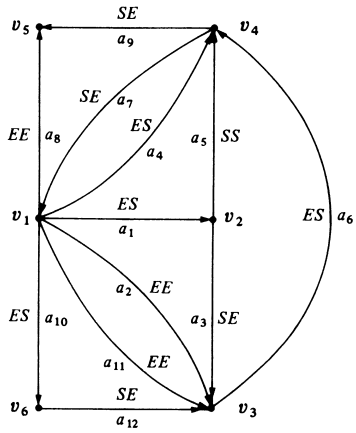


Figure 2. Subgraph representation of 'wait' relations between resources in a task.

(2) A directed graph G_i representing the wait relation in the ith task is obtained as a union ($G_i = \bigcup_j G_{ij}$) of subgraph G_{ij} (Fig. 3).

(3) There are some tasks which cannot be represented with only one Gantt Chart because they have more than one path. For such a task, graph G_i is obtained in the following manner. First draw the Gantt Charts and graphs for the individual paths. The union $G_i^{(p)}$ corresponds to the pth path. The graph G_i for all the paths is given as the union $G_i = \bigcup_p G_i^{(p)}$.



$$\mathbb{S}_i = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & a_{10} & a_{11} & a_{12} \end{array} \\ \begin{bmatrix} 1 & 1 & & 1 & & & -1 & 1 & & 1 & 1 & \\ -1 & & 1 & & 1 & & & & & & & \\ & -1 & -1 & & & 1 & & & & & -1 & -1 \\ & & & -1 & -1 & -1 & 1 & & 1 & & & \\ & & & & & & & -1 & -1 & & & \\ & & & & & & & & & -1 & & 1 \end{bmatrix} \end{array}$$

Figure 3. Directed graph representation of 'wait' relations between resources in a task.

(4) The directed graph $G = (V, A, \mathbb{S})$, which represents wait relations between resources in the system, is represented as $G = \bigcup_i G_i$, the union of G_i.

3.2 Non-propagation matrix and non-concurrency matrix

In order to analyse the propagation of wait relations, the non-propagation matrix and non-concurrency matrix are introduced.

First, the definition of concurrency between arcs is given.

Definition 5. Concurrency between arcs

If there is a possibility that the wait relations represented by arcs a_l and a_m occur at the same time, it is said that there is a concurrency between a_l and a_m.

Based on the above definition, the non-concurrency matrix is defined as follows:

Definition 6. Non-concurrency matrix

Let $U = \{(T_i, T_{i'})\}\,(i \neq i')$ be a set of pairs of tasks which are never activated simultaneously. The non-concurrency matrix is defined as $\mathbb{P}$, where the (l, m) element of $\mathbb{P}$ is

$$p_{lm} = \begin{cases} 1: \text{ If } a_l \in G_i,\ a_m \in G_{i'}\,(i \neq i') \text{ and } (T_i,\ T_{i'}) \in U, \\ \quad \text{or } a_l,\ a_m \in G_i \text{ and } a_l,\ a_m \notin G_{ij}; \\ 0: \text{ If otherwise.} \end{cases}$$

Next, the non-propagation matrix is defined, representing the propagation of the wait relations between resources.

Definition 7. Non-propagation matrix

The non-propagation matrix is defined as $\mathbb{Q}$ where the (l, m) element of $\mathbb{Q}$ is

$$q_{lm} = \begin{cases} 1: \text{ If arc } a_l \text{ is incident into vertex } v_k, \text{ arc } a_m \\ \quad \text{is incident out of vertex } v_k \text{ and } a_l \text{ is an} \\ \quad \text{ES or SS arc, } a_m \text{ is an SE or SS arc;} \\ 0: \text{ If otherwise.} \end{cases}$$

Both the non-concurrency matrix and the non-propagation matrix defined above have the same properties in the sense that they describe the conditions for non-propagation of wait relations in graph G. Thus, a 'restriction matrix' $\mathbb{R} = (r_{lm})$ is obtained as follows:

$$\mathbb{R} = \mathbb{P} + \mathbb{Q}, \quad r_{lm} = p_{lm} \vee q_{lm}.$$

4. SUFFICIENT CONDITIONS FOR SYSTEM DEADLOCK PREVENTION

In this section, sufficient conditions for system deadlock prevention are discussed for two cases. First, where there is only one resource of each type. Second where there is more than one resource of each type.

4.1 Sufficient conditions for system deadlock prevention where there is only one resource of each type

Let $\mathbb{C} = (c_{ij})$ be a directed circuit matrix in graph $G = (V, A, \mathbb{S})$ ($c_{ij} = 1$ if a_j belongs to the ith circuit in G, $c_{ij} = 0$ if otherwise). Then the following theorem holds true where there is only one resource of each type.

Theorem 1

Given graph $G = (V, A, \mathbb{S})$ and restriction matrix $\mathbb{R}$, sufficient conditions for system deadlock prevention are (1) no directed circuit exists in graph G, or (2) if directed circuits exist, $\mathbb{C}_k \times \mathbb{R} \times \mathbb{C}_k^T \neq 0$ for each vector $\mathbb{C}_k$ of directed circuit matrix $\mathbb{C}$ ($\mathbb{C}_k^T$: column vector of $\mathbb{C}_k$).

Proof

First, if no directed circuit exists, then there is no circular waiting. Thus conditions for the occurrence of system deadlock defined in Definition 2 are not satisfied. Next, from Definitions 5 and 6 it can be concluded that even if there are directed circuits $\mathbb{C}_k\,(k = 1, 2, \ldots, K)$, if $\mathbb{C}_k \times \mathbb{R} \times \mathbb{C}_k^T \neq 0$ for each circuit $\mathbb{C}_k$, then there is no possibility that wait relations will occur simultaneously in each circuit. Using Definition 7, there is no possibility that wait relations will propagate through each circuit. Thus, conditions for the occurrence of a system deadlock defined in Definition 2 are not satisfied (*end of proof*).

4.2 Sufficient conditions for system deadlock prevention in the case where there is more than one resource of each type

Let $\mathbb{E} = (e^{(1)}, e^{(2)}, \ldots, e^{(M)})$, showing the number of resources available in the system; $e^{(m)}$ = number of type m resources ($m = 1, 2, \ldots, M$). Let $\mathbb{U}_{ij} = (u^{(1)}, u^{(2)}, \ldots, u^{(M)})$, showing the allocation of resources among tasks; $u^{(m)}$ = number of type m resources allocated to task T_i at

time points t_j. Let $\mathbb{r}_{ij} = (r^{(1)}, r^{(2)}, \ldots, r^{(M)})$ showing requests for system resources; $r^{(m)}$ = number of type m resources requested at time point t_j by task T_i. The $\mathbb{u}_{ij}$ and $\mathbb{r}_{ij}$ can easily be obtained from the Gantt Chart of task T_i (see section 3.1).

The next theorem holds where there is more than one resource of each type.

Theorem 2

When graph G, restriction matrix $\mathbb{R}$, and the number of resources of each type $\mathbb{E}$, $\mathbb{u}_{ij}$, $\mathbb{r}_{ij}$ are given, the sufficient conditions for system deadlock prevention are: (1) G and $\mathbb{R}$ meet the conditions in Theorem 1, or (2) if G and $\mathbb{R}$ do not meet the conditions in Theorem 1 for directed circuits $\mathbb{c}_1, \mathbb{c}_2, \ldots, \mathbb{c}_k, \ldots, \mathbb{c}_K$, there is a full sequence $\alpha(k)$, where

$$\exists_m\, 0 < r_k^{(m)} \leqslant e^{(m)} - u_{\alpha(k)}^{(m)} \quad k = 1, 2, \ldots, K,$$

where

$$\mathbb{u}_{\alpha(k)} = \sum_{\{\bigcup_{\alpha(n) < \alpha(k)} ij | A_{ij} \cap \overline{W}_n \neq \varnothing\}} \mathbb{u}_{ij}$$

$$= (u_{\alpha(k)}^{(1)}, u_{\alpha(k)}^{(2)}, \ldots, u_{\alpha(k)}^{(M)})$$

$$\mathbb{r}_k = \sum_{\{ij | A_{ij} \cap \overline{W}_k \neq \varnothing\}} r_{ij}$$

$$= (r_k^{(1)}, r_k^{(2)}, \ldots, r_k^{(M)})$$

$$\overline{W}_k = \{a_l | c_{lk} = 1\}.$$

Proof

This theorem is proved only for the case in which there are directed circuits $\mathbb{c}_1, \mathbb{c}_2, \ldots, \mathbb{c}_K$ that do not meet the condition in Theorem 1.

First, suppose all the wait relations described by $\mathbb{c}_1, \mathbb{c}_2, \ldots, \mathbb{c}_K$ hold true simultaneously. Then the total number of resources allocated to the tasks in the circuits is

$$\sum_{\{\bigcup_{k=1}^{K} ij | A_{ij} \cap \overline{W}_k \neq \varnothing\}} \mathbb{u}_{ij}$$

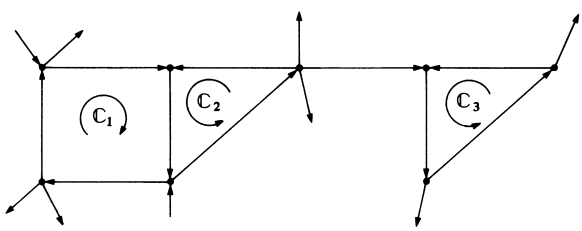


Figure 4. Situation when 'wait' relations in circuits occur simultaneously.

(Fig. 4) and the maximum number of resources remaining free or becoming free within a finite time is

$$\sum_{\{\bigcup_{k=1}^{K} ij | A_{ij} \cap \overline{W}_k \neq \varnothing\}} \mathbb{u}_{ij}.$$

On the other hand, $\mathbb{r}_k$ describes the number of resources for which tasks in $\mathbb{c}_k$ are waiting to be released. First k, is selected such that $\alpha(k_1)$ is the largest in $\alpha(k)$. Then, from the condition of the full sequence, there exists m such that

$$0 < r_{k_1}^{(m)} \leqslant e^{(m)} - u_{\alpha(k_1)}^{(m)}$$

Thus, the wait relation corresponding to $\mathbb{c}_{k_1}$ is dissolved. When this happens the number of resources allocated to the tasks in the circuits decreases to

$$\sum_{\{\bigcup_{\alpha(n) < \alpha(k_1)} ij | A_{ij} \cap \overline{W}_n \neq \varnothing\}} \mathbb{u}_{ij}.$$

Next, k_2 is selected such that $\alpha(k_2)$ is the next largest in $\alpha(k)$. Then, from $\{n | \alpha(n) < \alpha(k_1)\} = \{n | \alpha(n) \leqslant \alpha(k_2)\}$ and the condition of the full sequence, there exists m such that $0 < r_{k_2}^{(m)} \leqslant e^{(m)} - u_{\alpha(k_2)}^{(m)}$. Thus, the wait relation corresponding to $\mathbb{c}_{k_2}$ is also dissolved. In a similar way, the remaining wait relations are also dissolved (*end of proof*).

Next, an algorithm is introduced to check whether a full sequence $\alpha(k)$ exists.

Step 1

Let $\delta_0 = \{1, 2, \ldots, K\}$ be a set of subscripts of directed circuit $\mathbb{c}_k$. Let $\delta_0, \delta_1, \delta_2, \ldots$ be the sequence of δ.

Step 2

Examine to see whether m exists such that $r_k^{(m)} \leqslant e^{(m)} - u_{\delta_h}^{(m)}$ holds true for all $k \in \delta_h$, where

$$\mathbb{u}_{\delta_h} = \sum_{\{\bigcup_{k' \in \delta_h} ij | A_{ij} \cap \overline{W}_{k'} \neq \varnothing\}} \mathbb{u}_{ij}.$$

If such a k does not exist then neither does $\alpha(k)$, and there exists the possibility of system deadlock. If such ks do exist then let the set of those ks be $\{k\}$ and proceed to Step 3.

Step 3

Let $\delta_{h+1} = \delta_h - \{k\}$. If $\delta_{h+1} = \varnothing$ then proceed to Step 4, otherwise repeat Step 2.

Step 4

From the sequence $\delta_0, \delta_1, \ldots, \delta_{h'}$, resulting from Step 3, $\alpha(k)$ can be obtained as follows. First, assign in sequence the numbers K to $|\delta_1| + 1$ to $\alpha(k)$ for $k \in \delta_0 - \delta_1$.

Next, assign in sequence the numbers $|\delta_1|$ to $|\delta_2| + 1$ to $\alpha(k)$ for $k \in \delta_1 - \delta_2$. Repeat these assignments until the numbers $|\delta_{h'}|$ to 1 are assigned to $\alpha(k)$ for $k \in \delta_{h'}$, where $|\delta_h|$ is the number of elements of set δ_h.

5. RELAXATION OF THE CONDITIONS FOR SYSTEM DEADLOCK PREVENTION

Conditions in Theorem 2 were discussed assuming that $\mathbb{c}_1, \mathbb{c}_2, \ldots, \mathbb{c}_K$ may occur simultaneously. However, there are cases in which $\mathbb{c}_1, \mathbb{c}_2, \ldots, \mathbb{c}_K$ do not occur simultaneously due to non-concurrency between arcs (Definition 6). When $\mathbb{c}_1, \mathbb{c}_2, \ldots, \mathbb{c}_K$ do not occur simultaneously, the maximum number of resources allocated to tasks in the circuits for a given time becomes smaller than

$$\sum_{\{\bigcup_{k=1}^{K} ij | A_{ij} \cap \overline{W}_k \neq \varnothing\}} \mathbb{u}_{ij}.$$

Accordingly, relaxation of the conditions is possible. If there is a possibility that the circuits belonging to some of the subsets of $\mathbb{c}_1, \mathbb{c}_2, \ldots, \mathbb{c}_K$ occur simultaneously, then the conditions in Theorem 2 need be applied only to these subsets.

Whether or not $\mathbb{c}_1, \mathbb{c}_2, \ldots, \mathbb{c}_K$ occur simultaneously can be checked by the non-concurrency matrix $\mathbb{P}$ introduced in section 3.2. From the above, it is clear that the following proposition holds true.

Proposition 1

Let $C = \{\mathbb{c}_1, \mathbb{c}_2, \ldots, \mathbb{c}_K\}$. Let $C^{(l)}$ be a subset of C. If $\mathbb{C}^{(l)} \times \mathbb{P} \times \mathbb{C}^{(l)T} \neq 0$, then there will be no case in which any combination of circuits in $C^{(l)}$, where $\mathbb{C}^{(l)} = \bigvee_k \mathbb{c}_k | \mathbb{c}_k \in C^{(l)}$, occurs simultaneously.

From this proposition it is clear that the following theorem holds.

Theorem 3

Where a full sequence $\alpha(k)$ which meets the conditions in Theorem 2 does not exist, the sufficient conditions for system deadlock prevention are as follows. There exists a full sequence $\alpha^{(l)}(k)$ which meets the following conditions for each subset $C^{(l)} \subseteq C$ such that $\mathbb{C}^{(l)} \times \mathbb{P} \times \mathbb{C}^{(l)T} = 0$,

$$\exists_m \; 0 < r_k^{(m)} \leqslant e^{(m)} - u^{(m)}_{\alpha^{(l)}(k)} \quad k = 1, 2, \ldots, K^{(l)},$$

where

$$\mathbb{u}_{\alpha^{(l)}(k)} = \sum_{\{ij | A_{ij} \cap \hat{W}_n^{(l)} \neq \varnothing\}} \mathbb{u}_{ij} = (u^{(1)}_{\alpha^{(l)}(k)}, u^{(2)}_{\alpha^{(l)}(k)}, \ldots, u^{(M)}_{\alpha^{(l)}(k)})$$

and where the subscripts of each element in $C^{(l)}$ are reassigned as

$$C^{(l)} = \{\hat{\mathbb{c}}_1^{(l)}, \hat{\mathbb{c}}_2^{(l)}, \ldots, \hat{\mathbb{c}}_{K^{(l)}}^{(l)}\}$$

and $W_n^{(l)}$ is a set of arcs which make up $\hat{\mathbb{c}}_n^{(l)}$.

Proof

It is obvious from the proof of Theorem 2 and Proposition 1.

Corollary 3.1.

If there exists a full sequence $\alpha^{(l)}(k)$, where

$$\exists_m \; 0 < r_k^{(m)} \leqslant e^{(m)} - u^{(m)}_{\alpha^{(l)}(k)} \quad k = 1, 2, \ldots, K^{(l)}$$

for the subset $C^{(l)}$ such that $\mathbb{C}^{(l)} \times \mathbb{P} \times \mathbb{C}^{(l)T} = 0$, then there exists a full sequence $\alpha^{(l')}(k)$, where

$$\exists_m \; 0 < r_k^{(m)} \leqslant e^{(m)} - u^{(m)}_{\alpha^{(l')}(k)} \quad k = 1, 2, \ldots, K^{(l')}$$

for all subsets such that $C^{(l')} \subset C^{(l)}$ except when $C^{(l')} = \varnothing$.

From Corollary 3.1 it can be concluded that it is only necessary to check whether or not there exist $\alpha(k)$s for $\{C^{(l)}_{\max}\}$, the set 'maximal subset' of C, rather than checking for all subsets of C that meet $\mathbb{C}^{(l)} \times \mathbb{P} \times \mathbb{C}^{(l)T} = 0$. Here, $C^{(l)}_{\max}$, the maximal subset of C is defined as follows:

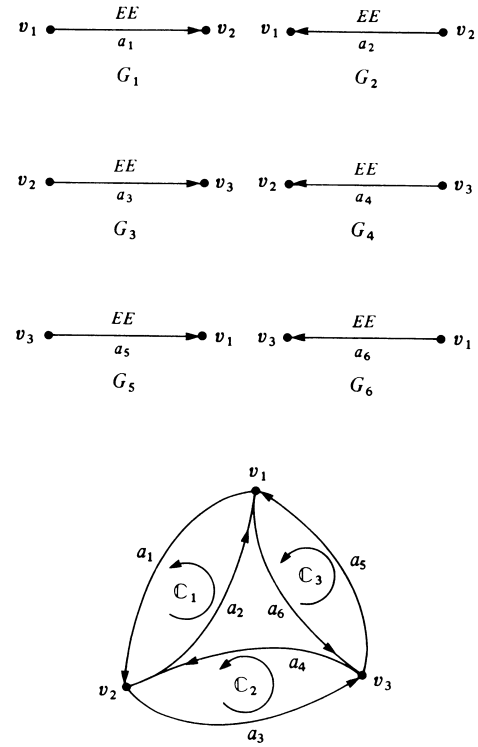
(1) $\mathbb{C}^{(l)} \times \mathbb{P} \times \mathbb{C}^{(l)T} = 0$

(2) $\forall_i \mathbb{c}_i \notin C^{(l)}_{\max} : (\mathbb{C}^{(l)}_{\max} + \mathbb{c}_i) \times \mathbb{P} \times (\mathbb{C}^{(l)}_{\max} + \mathbb{c}_i)^T \neq 0,$

where $\mathbb{C}^{(l)}_{\max} = \bigvee_j \mathbb{c}_j | \mathbb{c}_j \in C^{(l)}_{\max}$.

The same algorithm as in Theorem 2 can be applied to check whether or not a full sequence exists for each maximal subset of C.

The relaxation of conditions of Theorem 3 is shown by a simple example. As is seen in Fig. 5, in a system consisting of six tasks $T_1 \sim T_6$, their wait relations are shown by graph $G_1 \sim G_6$. Allocation of resources to each



$$\mathbb{P} = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{array} \begin{array}{c} \begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

$\mathbb{c}_1 = (1\ 1\ 0\ 0\ 0\ 0)$
$\mathbb{c}_2 = (0\ 0\ 1\ 1\ 0\ 0)$
$\mathbb{c}_3 = (0\ 0\ 0\ 0\ 1\ 1)$

Figure 5. An example of directed circuits in a system.

task is on an exclusive basis, and each type of resource allocated is counted as one resource. In graph G showing the wait relations of the system, $G = \sum_{i=1}^{\sigma} G_i$. Here tasks T_1 and T_3, T_3 and T_5, T_5 and T_1, T_2 and T_4, T_4 and T_6, and T_6 and T_2 are taken as never being activated simultaneously ($U = (T_1, T_3), (T_3, T_5), (T_1, T_5), (T_2, T_4), (T_4, T_6), (T_2, T_6)$: see Definition 6 in section 3.2). As seen in Fig. 5, three directed circuits, $\mathbb{c}_1$, $\mathbb{c}_2$ and $\mathbb{c}_3$ do not satisfy the conditions of Theorem 1 ($C = \{\mathbb{c}_1, \mathbb{c}_2, \mathbb{c}_3\}$). Since here

$$(\mathbb{c}_1 \vee \mathbb{c}_2 \vee \mathbb{c}_3) \times \mathbb{P} \times (\mathbb{c}_1 \vee \mathbb{c}_2 \vee \mathbb{c}_3)^T \neq 0$$

the three directed circuits $\mathfrak{c}_1$, $\mathfrak{c}_2$, $\mathfrak{c}_3$ are never activated simultaneously. Moreover, $\mathfrak{c}_1$ and $\mathfrak{c}_2$, $\mathfrak{c}_1$ and $\mathfrak{c}_3$, and $\mathfrak{c}_2$ and $\mathfrak{c}_3$ are likewise not activated simultaneously. In this example the maximal subsets of C are $C^{(1)}_{\max} = \{\mathfrak{c}_1\}$, $C^{(2)}_{\max} = \{\mathfrak{c}_2\}$, $C^{(3)}_{\max} = \{\mathfrak{c}_3\}$. For these three maximal subsets (in this example they are subsets consisting of only one element each), a check can be made of the conditions of Theorem 2. Supposing that the directed circuits $\mathfrak{c}_1$, $\mathfrak{c}_2$, $\mathfrak{c}_3$ are activated simultaneously, then the number of allocated resources would be

$$\begin{pmatrix} v_1 & v_2 & v_3 \\ 2, & 2, & 2 \end{pmatrix}$$

According to the conditions of Theorem 2, the minimum number of resources sufficient for deadlock prevention ($\mathbb{E}''$) is either

$$\begin{pmatrix} v_1 & v_2 & v_3 \\ 3, & 2, & 2 \end{pmatrix}, \text{ or } \begin{pmatrix} v_1 & v_2 & v_3 \\ 2, & 3, & 2 \end{pmatrix}, \text{ or } \begin{pmatrix} v_1 & v_2 & v_3 \\ 2, & 2, & 3 \end{pmatrix}.$$

However, according to the conditions of Theorem 3, since among directed circuits $\mathfrak{c}_1$, $\mathfrak{c}_2$, $\mathfrak{c}_3$ only one can be simultaneously activated, the number of allocated resources at the time of activation is either

$$\begin{pmatrix} v_1 & v_2 & v_3 \\ 1, & 1, & 0 \end{pmatrix} \text{ or } \begin{pmatrix} v_1 & v_2 & v_3 \\ 0, & 1, & 1 \end{pmatrix} \text{ or } \begin{pmatrix} v_1 & v_2 & v_3 \\ 1, & 0, & 1 \end{pmatrix}.$$

Here the necessary number of resources for release of $\mathfrak{c}_1$ is either

$$\begin{pmatrix} v_1 & v_2 & v_3 \\ 2, & 1, & 0 \end{pmatrix} \text{ or } \begin{pmatrix} v_1 & v_2 & v_3 \\ 1, & 2, & 0 \end{pmatrix}.$$

Likewise, for $\mathfrak{c}_2$ the number is either

$$\begin{pmatrix} v_1 & v_2 & v_3 \\ 0, & 2, & 1 \end{pmatrix} \text{ or } \begin{pmatrix} v_1 & v_2 & v_3 \\ 0, & 1, & 2 \end{pmatrix}$$

and for $\mathfrak{c}_3$

$$\begin{pmatrix} v_1 & v_2 & v_3 \\ 2, & 0, & 1 \end{pmatrix} \text{ or } \begin{pmatrix} v_1 & v_2 & v_3 \\ 1, & 0, & 2 \end{pmatrix}.$$

For satisfying the three conditions $C^{(1)}_{\max}$, $C^{(2)}_{\max}$, $C^{(3)}_{\max}$, the minimum number of resources $\mathbb{E}'''$ resulting in a sufficient system is either

$$\begin{pmatrix} v_1 & v_2 & v_3 \\ 2, & 2, & 1 \end{pmatrix}, \begin{pmatrix} v_1 & v_2 & v_3 \\ 1, & 2, & 2 \end{pmatrix} \text{ or } \begin{pmatrix} v_1 & v_2 & v_3 \\ 2, & 1, & 2 \end{pmatrix}.$$

This $\mathbb{E}'''$ is even smaller than the $\mathbb{E}''$ of Theorem 2, making it clear that the conditions of Theorem 3 have been relaxed compared to those of Theorem 2.

6. APPLICATION

In sections 4 and 5 the conditions for system deadlock prevention were obtained and the algorithms for checking deadlock possibility were developed by introducing three theorems.

In the practical development of application programs, it is recommended that Theorems 1, 2 and 3 be applied in sequence to reduce complexity. Not only can deadlock possibility be detected but its causes can be indicated by applying the algorithms derived from the theorems. Therefore, it is easy to eliminate the causes by using various approaches.[1, 2]

One approach is to increase the number of resources of the same type so as to satisfy the conditions of Theorem 3. Here care must be taken, in deciding which resource to increase, to minimise costs. First, let cost per unit of the type m resource of the system be $x^{(m)}$. As for the existence or non-existence of a full sequence $\alpha(k)$ referred to in the theorems, Step 2 of the decision algorithm described in section 4.2, if a k does not exist that satisfies:

$$\exists_m \; r^{(m)}_k \leqslant e^{(m)} - u^{(m)}_{\delta_h}$$

then the $q^{(m)}_k$ attained from the following:

$$q^{(m)}_k = r^{(m)}_k - e^{(m)} + u^{(m)}_{\delta_h}$$

indicates the number of resources m needed for the existence of a k that satisfies the abovementioned conditions. Next, type m_0 resource must be selected such that

$$\min_{m,k} x^{(m)} \cdot q^{(m)}_k = x^{(m_0)} \cdot q^{(m_0)}_{k_0}$$

the type m_0 resource increased in number, and the detection algorithm applied once again.

In existing process control computer systems, additions or modifications of application programs are often made, due to the extension of controlled plants or the enhancement of control methods. In these cases, the detection procedure for the possibility of deadlock must be carried out all over again. When a deadlock possibility is detected, its causes must be eliminated.

7. CONCLUSIONS

Using four predetermined application program parameters, a directed graph model representing the usage of common resources by tasks and a restriction matrix model were presented. Sufficient conditions for system deadlock prevention were derived, and algorithms for deciding whether or not the models meet the conditions were presented. It is easy to remove causes from the system because the algorithms can point out the causes for the possibility of deadlock.

The method proposed in this paper is especially useful in process control computer systems because the algorithms introduced do not have to be executed in real-time mode, and restrictions on the usage of common resources are applied only when a deadlock possibility is detected. Thus, additional side-effects on the responsiveness of the system will not arise, and a higher utilisation rate of common resources is guaranteed.

REFERENCES

1. E. G. Coffmann, M. J. Elphick and A. Shoshani, System deadlock. *Computing Surveys* **3** (2), 67–78 (1971).
2. J. W. Havender, Avoiding deadlock in multitasking systems. *IBM System Journal* **7** (2), 74–84 (1968).
3. J. W. Murphy, Resource allocation with interlock detection in a multitasking system. *Proceedings FJCC* **33**, 1169–1176 (1969).
4. A. Shoshani and E. G. Coffman, Detection and prevention of deadlock. *Proceedings, 4th Annual Princeton Conference on Information Sciences and Systems*, pp. 355–360 (1970).

5. D. Menasce and R. Muntz, Locking and deadlock detection in distributed data bases. *IEEE Trans. Software Eng.* **SE5** (3), 195–202 (1979).
6. V. D. Gligor and S. H. Shattuck, On deadlock detection in distributed systems. *IEEE Trans. Software Eng.* **SE-6** (5) 435–440 (1980).
7. A. N. Habermann, Prevention of system deadlock. *Comm. ACM* **12** (7), 373–377 (1969).
8. R. C. Holt, Comments on prevention of the system deadlocks. *Comm. ACM* **14** (1) 36–38 (1971).
9. A. Shoshani and E. G. Coffman, Sequencing tasks in multiproces system to avoid deadlocks. *Conference Record, 11th Annual Symposium on Switching and Automata Theory*, pp. 225–235 (1970).
10. L. H. Howard, Mixed solution for the deadlock problem. *Comm. ACM* **16** (7), 427–430 (1970).
11. D. B. Lomet, Subsystems of process with deadlock avoidance. *IEEE Trans. Software Eng.* **SE-6** (3), 297–304 (1980).
12. W. Clark, *The Gantt Chart* (3rd edn). Pitman, London (1952).