A Deadlock Avoidance Method for Concurrent Part Flows in Flexible Manufacturing Cell

Chang-Ouk Kim *
Kyung-Sik Kang **

Abstract

본 연구는 FMC(Flexible Manufacturing Cell)에 있어 상호 간섭이 없는 부품의 흐름을 제어하기 위한 교착제거방법(Deadlock Avoidance Method)에 대한 제안으로서, 이 방법이 주요한 장점은 첫째, FMC의 환경을 쉽게 적용시킬 수 있고 둘째, 공정상의 부품에 대한 dispatching 모듈이 협력하도록 설계되어 있다는 것이다. 교착제거방법은 두 개의 모듈로 구성되어 있는데 이는 순환발견알고리즘(A Cycle Detection Algorithm)과 원료주문정책으로서, 특이할 만한 사항은 순환발견알고리즘을 채용하더라도 원료주문정책이 잘못될 수 있기 때문에 교착제거방법에 있어서 원료주문정책은 매우 중요하다는 것이다. 이를 위하여 교착전의 N-step 교착을 제거하고, 발견할 수 있는 교착제거방법과 N-step의 사전주문정책을 제시하였다.

I. Introduction

A typical flexible manufacturing cell (FMC) consists of several machining centers, local buffers, and a material handling system. For increasing the performance and offsetting the high cost of capital investment, most FMCs are designed to simultaneously process several part types by assigning them to different machining centers according to their processing sequences. As a result, the complex moving patterns of parts cause interferences between parts for exclusive usage of resource.

* School of Industrial Engineering, Purdue University, West Lafayette, IN 47907, U.S.A., Tel: 317-746-2308, Fax: 317-494-1299, E-mail: kimco@gilbreth.ecn.purdue.e여
** Department of Industrial Engieering, Myong Ji university, san 38-2, Nam-Dong, Yong in, Kyunggi-do, 449-728, KOREA.

Therefore, development of a real-time control method which controls concurrent part flow without interference and yet achieves high utilization level of resource has been emerged as an important research issue.

The deadlock is one of the undesirable characteristics of FMC overlooked for many years in the research field of the automatic control of manufacturing system. This situation is occurred frequently during the operational stage of FMC because there are finite number of resources, and they may be requested by several concurrently moving parts in circular fashion. Furthermore, the possibility of deadlock increases with the density of parts in the system. Formally, deadlock in FMC is a situation in which two or more parts are in simultaneous wait state, each one waiting for one of others to release a resource before it can proceed.

A wrong idea to prevent deadlock is to operate a given FMC such that only one part type is allowed to be processed at a time (batch processing). This wrong idea arises from the viewpoint of machine-oriented deadlock. Most people consider only the machining sequences of various part types as the major factor of deadlock. Since only one part flow exists, deadlock may be looked avoidable. However, even if one cell consisting of a machine, a buffer, and a robot processes one part type, deadlock occurs frequently. For example, consider that part type 1 has the following processing sequence in the cell : the robot moves a new part from the buffer to the machine, the machine processes the part, and the robot moves the part from the machine to the buffer after finishing machining. Suppose a situation that one part is being processed by the machine and another part is being held by the robot waiting for the machine. Then further movement could not happen because the part on the machine cannot hold the robot which is already captured by the other part (circular wait). This example indicates that the development of deadlock resolution method must consider not only the machining sequences of part types but also the contention of other resources, such as material handling system, buffers, tools, pallets, and so on.

The objective of this research is to develop a deadlock avoidance algorithm which can resolve possible deadlocks in advance before the system enters the deadlock states, while preserving a high level of resource utilization. In Section 2, we discuss the deadlock problem of FMC in detail and related prior and current research works in this area. In Section 3, a resource-allocation graph which is used to represent part-resource relationship in a FMC is introduced. Section 4 provides the deadlock avoidance method which consists of a graph theoretic deadlock detection algorithm and a resource request policy. Finally, Section 5 concludes this paper and indicates possible areas of future research work.

Ⅱ. Literature Survey

Computer Science community, deadlock has been continuously observed during the implementation stage of process control software. It becomes a critical obstacle to efficiently operate the software in terms of resource utilization. Thus, the research related to deadlock was initially started by the Computer Science researchers to resolve infinite waiting state of jobs in operating systems (Peterson and Silberschatz, 1985) and of transaction in database system (Bell and Grimson, 1992). As a result, the following necessary conditions were derived for deadlock to occur :

- *Mutual Exclusion* - Parts hold resources exclusively, making them unusable to other parts
- *Nonpreemption* - Resources are not snatched from one of the parts that are requesting them until they are released by a part holding them.
- *Blocking* - Parts which request unavailable resources must wait until the resources are released.
- *Circular Waiting* - Parts request resource held by other parts and at the same time hold resources requested by these same parts.

In particular, as commonly observed in the part flows in FMC, if each resource is assumed to have single capacity unit which implies a resource may be assigned to at most one distinguishable part, then the above four conditions become sufficient and necessary condition for deadlock (Peterson and Silberschatz 1985). At the design or operational stage of FMC, deadlock can be precluded by configuring FMC or devising a controller such a way that at least one of the four conditions is eliminated. But, due to technological limitation and material cost, most FMCs do no allow resource sharing and preemption of part. Therefore, resolution of either the blocking or the circular waiting is the only economic way not to fall into deadlock state. According to the point in time of deadlock resolution at the operational stage of FMC, deadlock resolution method can be classified into the following three methods.

2.1. Deadlock prevention method

The deadlock prevention method keeps the system from deadlock by disallowing entry into a unsafe state from which future deadlock is inevitable. Most deadlock prevention methods have been developed by relying on Petri net. Viswanadham *et al.* (1990) applied the reachability graph of Petri net to construct the deadlock prevention policy of a given FMC. However, the exhaustive state enumeration of the reachability graph of a medium or huge size FMC leads to excessive computing time for the enumeration. Thus, their method seems to be impractical to be applied to real FMCs. For the real-time control of automated guided vehicle (AGV), Kim and Tanchoco (1991)

devised a deadlock prevention algorithm by introducing a time-window mechanism.

2.2. Deadlock detection/recovery method

In some sence, the deadlock prevention method is a static policy because it does not take account of dynamic information of system status to circumvent deadlock, resulting in a poor resource utilization. To obtain much higher performance level, the deadlock detection and recovery is a preferred alternative in some cases, such as databases and operating systems (Peterson and Silberschatz 1985, Bell and Grimson 1992) where it is not critical and expensive to abort one of circular-waiting jobs and restart it. Wysk *et al.* (1991) applied a symbol matrix and a string manipulation algorithm to detect a deadlock and recover it with special buffers which are reserved for breaking deadlock.

2.3. Deadlock avoidance method

The deadlock avoidance method is somewhat in the midst of the deadlock prevention method and the deadlock detection/recovery method in terms of deadlock resolution time. Like the deadlock detection/recovery method, it monitors and uses dynamic information of system status to avoid deadlock. But the deadlock avoidance method uses the system information to find out a way which leads the system not to enter deadlock state, while the deadlock detection/recovery method uses it to detect deadlock state. Meanwhile, it can be said that the deadlock avoidance methods is similar to the deadlock prevention method in the sense that both methods do not allow the system to be led to deadlock state. However, from the viewpoint of resource utilization, the deadlock avoidance method is better due to its control flexibility. Thus, among the three methods, the deadlock avoidance method is the most suitable method to be applied in FMC.

Petri net is a popular for deadlock avoidance in FMC. Viswanadham, et al. (1990) used the reachability graph of Petri net and a n-step lookahead approach for deadlock free resource allocation. However, they mentioned that infinite look-ahead is inevitable to avoid all possible deadlocks. Thus, their method can be only applied to small size systems. Banaszak and Krogh (1990) introduced the notion of restriction policy, one of the resource request policies by which some enabled transitions that may lead to deadlock condition are excluded during operational stage. The core of Hsieh and Chang's work (1994) is to use liveness condition. They developed a test procedure which can determine whether the system is live after an action is executed. Although the Petri net modules are successful to avoid deadlock in FMC, they needs professional skill to be constructed, and cannot be modified easily when the configurations of FMCs or the processing sequences of part types change dynamically.

Ⅲ. Resource Allocation Graph

The resource-allocation graph has been recognized a powerful model which describes deadlock precisely for many years in the area of multitasking operating systems (Peterson and Silberschatz 1985). This graph is a *time-varying directed bipartite graph* G(t)=(V, E) consisting of a set V(t)={1, 2, ..., $|V(t)|$} of nodes and a set $E(t) \subseteq V(t) \times V(t)$ of edges at time t. A pair $(u, v) \in E(t)$ is called an edge from u to v at time t. V(t) is partitioned into two subsets $P(t)=\{p_1, p_2, \ldots, p_n\}$, the set of parts being processed on the system at time t, and $R(t)=\{r_1, r_2, \ldots, r_m\}$ the set of resources available at time. Pictorially, each part $p_i, i=1,2,\ldots,n$ is represented as a circle and each resource $r_j, j=1,2,\ldots,m$ as a square. A *path* from u to v where $u, v \in V$ is a sequence of nodes, $v_0, v_1, \ldots, v_k$ such that $v_0=u$, $v_k=v$ and $(v_i, v_{i+1}) \in E$ for $0 \leq i < k$ where k is the length of the path. A *cycle* is a path from u to v and a graph is *acyclic* if it contains no cycles. Indegree of a node v which is denode by $indeg_{G(t)}(v) = |\{u_i(u, v) \in E\}|$, is the number of edges ending at v, and outdegree a node v, outdeg $outdeg_{G(t)}(v) = |\{u_i(v, u) \in E\}|$, is the number of edges starting at v.

There are two types of edges in the resource-allocation graph : request edge and allocation edge. The request edge, (p_i, r_j) is a directed edge from part p_i to resource r_j. It represents that part p_i requested resource r_j and is currently waiting for that resource. The allocation edge, (r_j, p_i) is a directed edge from resource r_j to part p_i, signifying that resource r_j has been allocated part p_i. Note that $outdeg_{G(t)}(r_j)=1, j=1,2,\ldots,m$. This implies that each resource must be held in a nonsharable mode. Also, no edge exists between parts or between resource. Figure 1-a shows an example of the resource-allocation graph.

In a given FMC, the dynamic change of part-resource relationship as the results of requests, allocations, and releases of resources are reflected in the resource-allocation graph by the insertions of request edges, the changes of the request edges to allocation edges, and the deletions of the allocation edges. Under the assumption that each resource has one capacity unit, deadlock can be predicted by checking whether an insertion of edge makes the resource-allocation graph cyclic. Therefore, resource allocation problem with deadlock avoidance can be considered as the problem of changing the resource-allocation graph while keeping it acyclic. Hereafter, it is assumed that each part is allowed to capture a resource only after currently holding resource is released. This is a common way of resource management. But each part is granted to request a resource anytime. Then, deadlock of FMC can be classified into two types :

request-oriented deadlock and allocation-oriented deadlock.
The request-oriented deadlock is the one that a request-edge makes deadlock. Figure 1-b show an example of the request-oriented deadlock where insertion of request edge (1, A) forms a cycle (1, A, 2, B, 3, C, 1). The allocation-oriented deadlock is caused by an allocation of a resource to a part. Figure 1-c show an allocation-oriented deadlock which occurs when request edge (1, A) change to allocation edge (A, 1), thereby creating a cycle (A, 1, B, 2, A). Therefore, request and allocation of a resource must be controlled carefully such that the system is deadlock free.

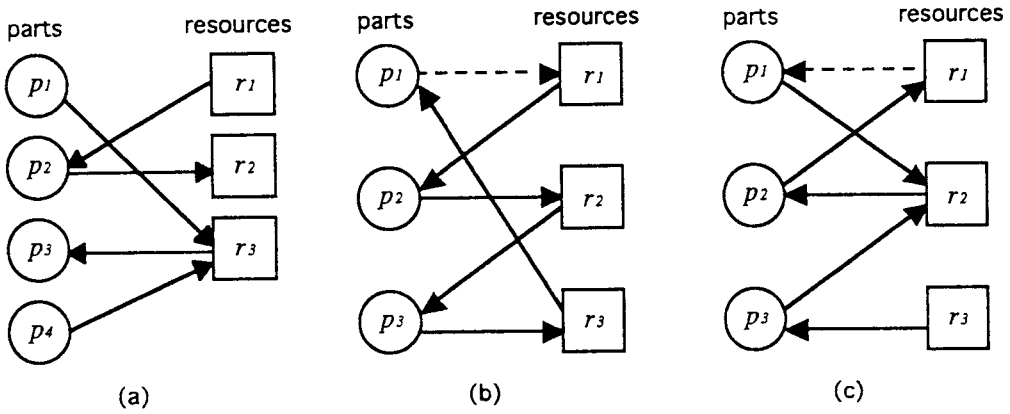


Figure 1. (a) acyclic resource-allocation graph, (b) request-oriented deadlock, and (c) allocation-oriented deadlock

Ⅳ. Deadlock A voidance Method

4.1 Cycle detection algorithm

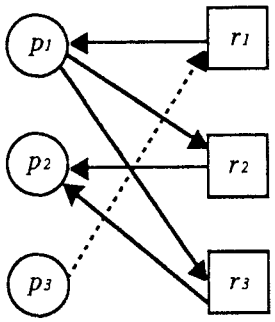
In this section, we investigate a cycle detection algorithm originally developed by Belik (1990). The cycle detection algorithm is based on a path matrix. The path matrix is a $|V| \times |V|$ matrix P, with

$$P(i,j) = \begin{cases} k & \textit{if there are } k \geq 1 \textit{ different paths from node } i \textit{ to node } j \\ 0 & \textit{if there is no path from node } i \textit{ to node } j \end{cases}$$

where $1 \leq i,\ j \leq |V|$

Figure 2-b shows the path matrix of a resource-allocation graph depicted in Figure 2-a. We define the column vector of P as $P(\_, j)$, the ith row vector as $P(i, \_)$, and the cross product between the column vector and the row vector as $P(\_, j) \oplus P(i, \_)$, which is again $|V| \times |V|$ matrix. Also, let I(j) denote identity a column vector of size

$|V|$ where each element has zero value except at element j, and $I(j)^T$ denote the transpose of $I(j)$.



(a)

	p_1	p_2	p_3	r_1	r_2	r_3
p_1	0	2	0	0	1	1
p_2	0	0	0	0	0	0
p_3	0	0	0	0	0	0
r_1	1	2	0	0	1	1
r_2	0	1	0	0	0	0
r_3	0	1	0	0	0	0

(b)

Figure 2. (a) a resource-allocation graph and (b) corresponding path matrix

In Figure 2-a, consider the insertion of an edge (p_3, r_1). Then the number of paths passing through the edge (p_3, r_1) is formed as follows : the number of paths reaching node p_3 and the number of paths starting from node r_1 correspond to column vector $P(\_, p_3)$ and row vector $P(r_1, \_)$, respectively. Therefore, the number of paths which passes through edge (p_3, r_1) can be produced by the cross product of $P(\_, p_3)$ and $P(r_1, \_)$. What we do not consider is the number of paths starting from p_3 or ending at r_1. It is derived by adding $I(p_3)$ to $P(\_, p_3)$ and $I(r_1)^T$ to $P(r_1, \_)$. Thus, of we denote $\Delta(p_3, r_1)$ as a matrix whose elements represent the number of paths either passing through edge (u, v) and the number of paths starting from u or ending at v, then $\Delta(p_3, r_1)$ is given by

$$\Delta(p_3 r_1) = (P(-, p_3) + I(p_3)) \otimes (P(r_1, \_) + I(r_1)^T) \quad (1)$$

By adding $\Delta(p_3, r_1)$ to path matrix F which is the one before inserting edge (p_3, r_1) in Figure 2-b, the number of paths being dynamically changed during the operational stage of FMC can be recorded. For example, as given in Figure 2-a, if edge (p_3, r_1) is inserted, $\Delta(p_3, r_1)$ is calculated by

$$\Delta(p_3, r_1) = \left[\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \right] \otimes [(1\ 2\ 0\ 0\ 1\ 1) + (0\ 0\ 0\ 1\ 0\ 0)]$$

$$= \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ r_1 \\ r_2 \\ r_3 \end{array} \begin{array}{c} \begin{array}{cccccc} p_1 & p_2 & p_3 & r_1 & r_2 & r_3 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Therefore, the new path matrix is given by

$$P^{new} = P^{old} + \Delta(p_3, r_1) \tag{2}$$

$$= \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ r_1 \\ r_2 \\ r_3 \end{array} \begin{array}{c} \begin{array}{cccccc} p_1 & p_2 & p_3 & r_1 & r_2 & r_3 \end{array} \\ \begin{bmatrix} 0 & 2 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 1 & 1 & 1 \\ 1 & 2 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

The problem that remains to be considered is to check whether insertion of an edge (p_3, r_1) makes the graph cyclic, which is equivalent to a deadlock state in FMC. When attempting to insert an edge (p_3, r_1) into the graph, it is enough to check whether path matrix element $P(p_3, r_1) > 0$, which means that there exists at least one path from node p_3 to node r_1. Thus, the insertion of edge (u, v) will make the graph cyclic. In the previous example, the FMC is safe from deadlock because $P^{new}(r_1, p_3) = 0$.

For an edge deletion, the same procedures are followed except the equation 2 where instead of the addition operation, $\Delta(r_1, p_3)$ is subtracted from P^{ola} .

The cycle detection algorithm is adaptable to the change of FMC configuration (installation or removal of resources) by adding or deleting new resource nodes in the resource allocation graph. The cycle detection algorithm also dose not use a prior knowledge of routing sequences of parts for resolving deadlock. This feature implies that the routing sequences cant be flexibly changed according to the state of FMC, which allows the dispatching controller to use a flexible routing control. Whereas, Petri-net based deadlock detection methods must reflect the routing sequences of parts on the Petri-net models to detect deadlock before FMC beings to operate

4.2. $N-step(N \geq 1)$ lookahead resource request policy

The resource request policy refers to a strategy that decides the point in time of requesting resources. It is of importance for the deadlock avoidance in FMC because (1) it affects the resource utilization and (2) FMC can even be led into deadlock although the deadlock detection algorithm is employed under an impertinent resource request policy. For example, consider 0-step lookahead (immediate) resource request policy under which every part requests the next resource just before it releases the currently holding resource. This policy will guarantee the highest resource utilization compared to other possible policies. Despite this advantage, however, the 0-step lookahead resource request policy may induce the request -oriented deadlock unless alternate routing sequence is provided. For instance, as shown the resource-allocation graph in Figure 3. if part p_1 requests resource r_1 after being processed in resource r_2, the cycle detection algorithm disallows part p_1 to make request edge (p_1, r_1) because the edge makes a cycle $(p_1, r_1, p_2, r_3, p_3, r_2, p_1)$. Therefore, part p_1 should wait until the parts p_2 and p_3 release resource r_2 and r_3 . But parts p_2 and p_3 also wait for part p_1 to release resource r_1. Consequently, albeit, there is no cycle, any part cannot move.

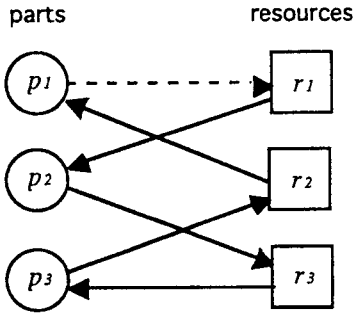


Figure 3. A request-oriented deadlock.

To avoid such a request-oriented deadlock, we propose a $N-step(N \geq 1)$ lookahead resource request policy. This policy always enforces every part to request the next resource and future N resources before releasing the currently holding resource. The range of N for part p_1 is $1 \leq N \leq M_1$ where M_1 is the number of required resources during the whole processing period of part p_1. Figure 4 shows the possible members of the $N-step$ lookahead resource request policy. The main distinction among the member policies is characterized by the resource utilization. As the lookahead step increases, the more the resource utilization decreases. Among the member policies, the 1-step lookahead resource request policy preserves the highest resource utilization.

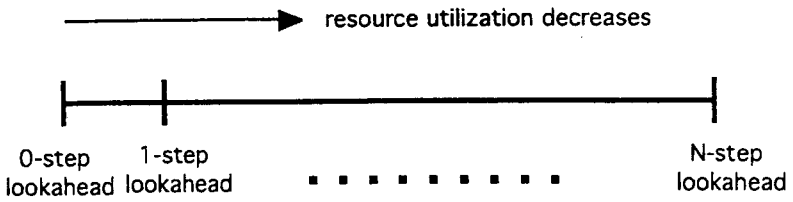


Figure 4. N-step(N≥1)lookahead resource request policy.

4.3 Deadlock avoidance method

The deadlock avoidance method is composed of two modules: the 1-step lookahead resource request policy and the cycle detection algorithem. Compared to other deadlock avoidance methods, this method is capable of interacting with a part dispatching controller. The following procedure illustrates the deadlock avoidance method.

Deadlock Avoidance Method(p_i, *next, signal*) {
  /* 1-step lookahead resource request policy */
  if(*next* == *1*)**then**
    insert request edges $p_i \rightarrow r^i_{next}$ and p_i and $p_i \rightarrow r^i_{next+1}$ in the resource-allocation graph
  else
    delete assignment edge $r_c^t \rightarrow p_i$ in the resource-allocation graph
    insert request edges $p_i \rightarrow r^i_{next+1}$ in the resource-allocation graph
  end if
/* the cycle detection method */
  calculate new path matrix using equation (1) and (2)
  if(r^i_{next} is idle and P(r^i_{next}, p_i))**then**
    change $p_i \rightarrow r^i_{next}$ to $r^i_{next} \rightarrow p_i$
    signal=yes
  else
    signal=no
  end if
  return*(signal)*
}

In the above procedure, the variables *next* and r_j^i are symbols to represent the next processing stage of part p_i and the resource require at the jth processing stage of part p_i, respectively. Also, the *signal* is a variable to indicate whether deadlock occurs or not. Figure 5 depicts an integration model of the deadlock avoidance controller and a dispatching controller. The integration model follows a client-server mechanism(Berson, 1992) where the dispatching controller acts as a client and the deadlock avoidance controller as a server. Whenever the processing of a part is finished, the dispatching controller refers to the deadlock avoidance controller for the deadlock possibility which may be caused by the request of the next resource.

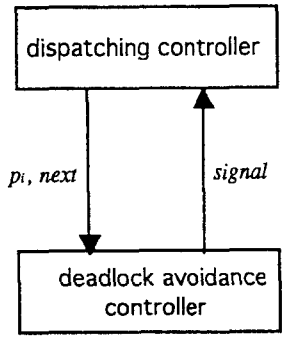


Figure 5. Integration of the deadlock avoidance controller and a dispatching controller.

V. Conclusion

Throughout this research, a deadlock avoidance method for FMC is proposed. Compared to other methods, the deadlock avoidance method is simple and can be easily implemented. Additionally, it is scalable in the sense that when a new resource is introduced such as the purchasing of a new matching center, the deadlock avoidance method can be still used without modification except the insertion of a node representing the resource in the resource-allocation graph and the path matrix. For future research, it is considerable to develop a full set of shop-floor controller which includes a scheduler, a dispatcher, a monitor, and a deadlock avoidance controller.

References

BANASZAK, A.Z., AND KROGH, B. H., Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. *IEEE transactions on robotics and automation,* ***6,*** 724-734.

BAUER, A., BOWDEN, R., BROWNE, J.,DUGGAN, J., AND LYONS., G., 1991, *shop floor control systems: from design to implementation*(Chapman and Hall).

BELIK, F., 1990, An efficient deadlock avoidance technique.*IEEE transactions on computer,* **39,** 882-888.

BELL DAVID, and ,GRIMSON J. 1992, *Distributed database systems* (Addison-Wesley).

BERSON, A., 1992, *Client/Server Architecture*(McGraw-Hill).

COFFMAN, E. G., ELPHICK, M. J., AND SHOSHANI, A., 1971, System deadlocks, *ACM Computing surveys,***3**, 67-78.

HOLT, R. C.,1971, Comments on prevention of system deadlocks, Communication of the ACM, **14**, 36-38.

HSIEH, F. S.,AND CHANG, S. C., 1994, Dispatching-driven deadlock avoidance controller synthesis for flexible manufacturing system. *IEEE transactions on robotics and automation,* **10**, 196-209

KIM, C. W.,AND TANCHOCO, J. M. A., 1991, Conflict-free shortest-time bidirectional AGV routing. *International Journal of Production Research,***29,**2377-2391.

KUNDU, S.,AND AKYILDIZ, I.F., 1989,Deadlock free buffer allocation in closed queueing networks.*Queueing Systems,* **4**,47-56.

PETERSON, J.L., AND SILBERSCHATZ, A., 1985, *Operating System Concepts.*(Reading, MA:Addison-Wesley).

SURI, R., AND STECKE, K. E.(editors), 1989, *Proceedings of the third ORSA/TIMS conference on flexible manufacturing systems: operations research model and applications,*(FMCterdam; New York: Elisevier).

VISW ANADHAM, N., AND NATAHARI, Y., 1992, *Performance Modeling of Automated Manufacturing Systems*(Englewood Cliffs, NJ:Prentice-Hall).

VISWANADHAM, N., NARAHARI, Y., AND JOHNSON, T. L.,1990, Deadlock prevention and deadlock avoidance in flexible manufacturing systems using petri net models. *IEEE transactions on robotics and automation,* **6**, 713-723.

WYSK, R. A., YANG, B. N., AND JOSHI, S., 1991, A detection of deadlocks in flexible manufacturing cells. *IEEE transactions on robotics and automation,* **7**,853-859

WYSK, R. A., YANG, N. S., AND JOSHI, S., 1994 Resolution of deadlocks in flexible manufacturing systems: avoidence and recovery approaches. *Journal of Manufacturing Systems,* **13**, 128-138.