MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Computation Structures Group Memo No. 45

Coordinated Sharing of Resources in Asynchronous Systems

Prakash Hebalkar

Note: The contents of this memo represent a proposal for doctoral research work submitted to the Department of Electrical Engineering on January 4, 1970.

January 1970

Introduction

A serious problem which arises in systems wherein two or more simultaneously active but asynchronous users share a limited amount of resources in an uncoordinated manner is that of deadlock. Deadlock is the situation in which the unallocated resources are inadequate to fulfill the needs of any of the users, while the users are unable to release the resources already allocated to them. An important characteristic of deadlock is that the users are held up indefinitely. If the users can in fact release resources allocated to them, the deadlock can clearly be resolved, so that the deadlock is really a pseudo-deadlock. As an example of the occurrence of deadlock consider a system with two units of a resource and two processes each of which eventually requires both the units of resource for completion. Suppose both the processes start simultaneously and request one unit of resource each, which they are assigned. Clearly the two processes are deadlocked and can only proceed to the point where each needs the other unit of resource.

Deadlocks can be avoided by coordination of the use (allocation) of resources. The coordination is quite rigid when the processes do not cooperate with the allocator. In particular if no information about usage of resources is provided by the users, coordination with a view to avoiding deadlock requires that the users be given access to the resources sequentially, i.e. one at a time. The coordination can be less restrictive when the users cooperate with the allocator and in a sense the greater the degree of cooperation (as measured by the amount of information about resource usage provided) the less heavy-handed the coordination has to be.

For example, suppose the users provide just a little information about usage, viz. the maximum amount needed. A simple way to avoid deadlock is to pick a user, put aside an amount of resources equal to his maximum demand, then if any resources are left over put aside some other user's maximum need and so on until the remaining resources are zero or insufficient for any user not yet taken care of. Then the users are allowed to proceed until some user finishes, when an unserved user is again sought, and so on until they all finish. When users do not use the maximum amount all the time, and this is frequently the case, it is clear that resources are lying idle in this kind of scheme. Fortunately, it does not have to be this way. Habermann [1] has shown that there is a better scheme of allocation that keeps down the amount of unused resources, albeit by requiring even users who are partially along to occasionally wait for some time. When the user programmes last for a long time and require only a small fraction of their peak requirement for most of the time, the latter scheme results in smaller total waiting times in general. Furthermore, it can in some sense do no worse than the first scheme even if the conditions in the previous statement did not hold. This will become clear in the brief discussion of his work which follows.

Habermann investigated the following situation: There are n users active at a time, each with a (pre-specified) maximum resource requirement $\underline{m}^i$. Each user makes requests periodically (the different users do so asynchronously) for additional amounts of resources. A request may be granted immediately or after an indefinite but finite delay. No resources are allocated unless requested for. Each user keeps all allocated resources until he finishes,

whereupon he returns them all. It will be noted that resource requirements are indicated by vectors i.e. there are requirements for each of say p kinds of resources. At any instant the system of users is in an allocation state defined by the various amounts of resources allocated to each user ($\underline{a}^i$) and an unused resource pool $\underline{u}$. An allocation state is said to be "safe" if all users can finish within a finite time (i.e. no deadlock can occur). If $\underline{c}$ be the resource capacity of the system, it is shown that an allocation state is safe if there exist indices $i_1, i_2, \ldots i_n$ (each i identifying a user and hence also distinct) such that

$$
\begin{aligned}
\underline{m}^{i_1} - \underline{a}^{i_1} &\leq \underline{u} \\
\underline{m}^{i_2} - \underline{a}^{i_2} &\leq \underline{u} + \underline{a}^{i_1} \qquad \ldots\ldots \text{I} \\
&\vdots \\
\underline{m}^{i_n} - \underline{a}^{i_n} &\leq \underline{u} + \sum_{j=i_1}^{i_{n-1}} \underline{a}^j
\end{aligned}
$$

i.e. if there exists an order in which the users can finish (user i_1, when he finishes, releases all allocated resources making it possible for user i_2 to finish and so on until all can finish). It should be noted that this is a possible worst-case order in which users can finish. Several such orders may exist for a given allocation state. Also the users do not in reality have to finish in that order. If at some instant some user (say the i_p^{th}) makes a request for additional resources, the allocator allocates them only if the allocation state that could result if the request is granted is still safe (i.e. if a worst case order of completion, not necessarily the same as above, exists), otherwise the request is help up until adequate resources are released by users who finish up.

It will be noticed that for any allocation state if an order exists satisfying I then a canonical order exists in which the same condition is still satisfied but the left hand sides of the inequalities are in non-decreasing order. Thus the unused resources $\underline{u}$ need only be as large as the smallest of the unsatisfied demands.

It is frequently the case that users can predict more than just the maximum amounts of each kind of resource that they will use. The following example which is inspired by Habermann's banking example [1] illustrates this. Consider a construction equipment rental company which rents equipment of all kinds to building construction firms. In view of the vast amounts of money invested in buildings that are incomplete, contractors are anxious to finish buildings as fast as possible. Consequently they are anxious that they not be help up indefinitely in the completion of work in process. However, the rental costs of equipment are so high that no contractor can rent at the start all the equipment he may need from ground-breaking time to ribbon-cutting time. Each contractor thus rents equipment only as needed and immediately returns any equipment he cannot use for some time (say a day or more), perhaps not any more (for this project). The rental firm of course tries all the time to rent out as much equipment as possible to the contractors. Each contractor knows that he will need at most x tower cranes, y excavators, (x and y are some numbers) and so on. However he also knows that when he needs excavators most he does not need all the tower cranes and so on. In fact he has a fairly detailed idea of the phases through which the project will proceed and the amount of equipment he needs at any time.

he of course does not know how many rainy days there will be or when he may have strikes or accidents or a shortage of workers so that his schedule really cannot be translated into specific days and months of the calendar. It is to the contractor's advantage to give his schedule, such as it is, to the rental firm, for if thereby the rental firm can increase equipment rental-days, it may charge a smaller fee for ensuring that no firm gets held up indefinitely because the equipment has been spread thinly. Because of the fiercely competitive nature of the business no contractor is willing to return equipment temporarily, when he can use it, just because the rental firm made an incorrect decision in allocating the equipment to the users. The problem is, "How can the rental firm make allocation decisions so as not to default on its no-deadlock commitment and yet maximize its income in some sense?"

The problem above has an analogue in computing systems wherein multiprocessing takes place. The processes that are active at any time represent the users (the contractors in the previous example) while the various resources such as input-output devices, general purpose registers, active memory space, arithmetic processors etc. represent the resources that are in simultaneous demand. The objection is frequently raised that as these resources can be withdrawn arbitrarily by the system resource allocator, the concept of deadlock is entirely irrelevant. However, is the premise of this argument true? Is it meaningful for the system to deprive a process of an input-output device (say a tape reader) part-way into its activity and allocate it to another process albeit with a promise of returning it later? It would seem not. Active memory space, arithmetic processors, etc. seem a little

different. After all, in a multi-level memory system, core space, for instance, can be reallocated provided any alterable data is stored away in the slower memory for recall later (unalterable data can be recalled without such an intermediate storing away). However, while this does mean that deadlock in the strict sense cannot occur, it does not mean that a scheme that reduces such transfers is not of interest. For every transfer from a fast memory to slow memory and vice versa takes a very long time in terms of use of the fast memory and processor time, and in the interest of reducing response time it is advisable to keep the number of such transfers down. Thus the analysis of deadlocks and their prevention in computing systems is important.

A valid objection to the study of the problem stated in the example above is that detailed a prior knowledge of resource usage by processes is not always available. However, some knowledge if only conservative estimates can be obtained (as of use of input-output devices say) during compilation or provided by other means. Moreover the relevant question really is how such information can be used to advantage, particularly as the model for this situation that will be presented in the next section also models situations where only information about the maximum usage of resources is available.

The Model

Given that a user or process (the two will be used synonymously henceforth) can indicate a priori how his demand for resources will vary with time, the relevant question is what form this information should take. The choice should ensure ease of analysis of the chosen form and ensure that the form is a natural one for the specifier to provide the information in. For instance one form in which the information can be provided is as graphs of amount demanded against process-time as in figure 1 which shows

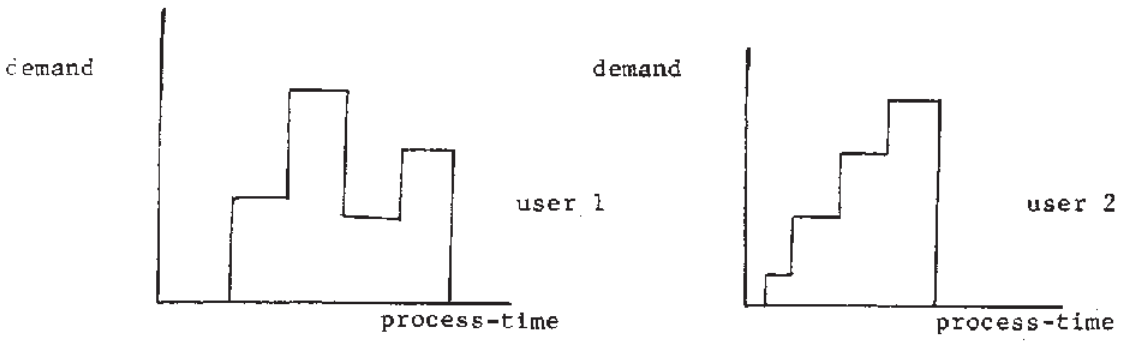

Figure 1

a situation with two users of one kind of resource. This form is not very convenient to analyse as the time axes are not comparable on account of the asynchronous nature of the processes. This lack of comparability of the graphs makes analysis difficult. Comparability of demand behaviour is possible in the proposed model which is presented next.

The example in section 1 has already given the reader hints about the model to be used. It pointed out that a contractor knows the phases that his project goes through and their order as well as his needs in each phase of the

project. Thus one comes upon a partially ordered directed graph for each user. The arcs of this graph represent the phases of the process and the nodes represent transitions between phases. The model will be developed as the discussion proceeds. For the present the following situation will be considered and the form of the model for it shown: there are m distinct processes which are completely asynchronous ("completely" means there are no points of synchronization); each process consists of some number (n_i for the i^{th} one) of phases which are completely ordered and there are non-negative requirements for the only kind of resource in the system. In the circumstances there are m distinct directed chains of arcs in the demand graph (or d-graph), one per process, each with one initial arc and one terminal arc. As a matter of convention the first and last arcs of each chain have a zero requirement of resource associated with them corresponding to the phases "process not yet begun" and "process finished". Also as a matter on convention adjacent arcs on a chain have to have distinct requirements associated with them, i.e. a transition between phases of a process is associated with a change in requirements. The resulting model is illustrated by figure 2. Figure 2 also illustrates some of the notation to be used. Thus associated with the j^{th} arc on the chain associated with user i (designated α_j^i) is the number $d(\alpha_j^i)$ which represents the quantity of resource needed in that phase of process i. The chain itself will be designated by X_i (chi-i). It will usually be the case that the resource is available in some units so that $d(\alpha_j^i)$ is an integer representing the number of such units. However this integer restriction is not crucial.

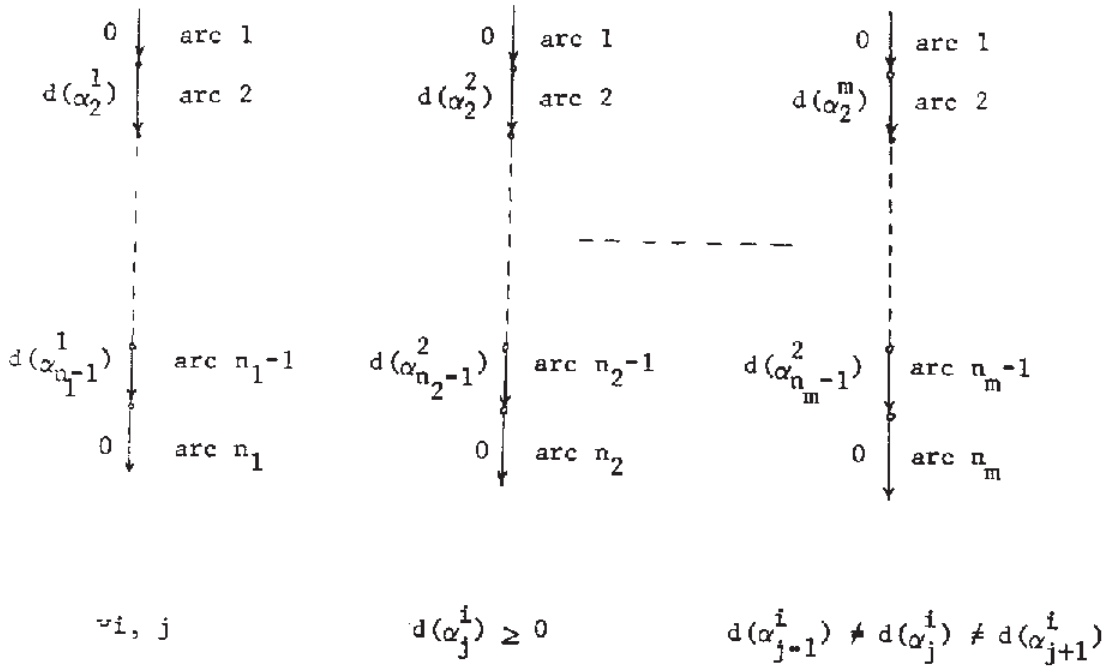

Figure 2

At any instant each process is in a certain phase and has a certain requirement for the resource. Since it is in the phase, that requirement must have been allocated too. Thus a composite of the phases of the m processes also defines the resource allocation state of the system. As each process can only be in one phase at a time, the states of the system correspond to cut-sets of the (composite) graph consisting of the m chains. Such cut-sets will be called slices (after Holt [2]). A slice will be designated by γ (sometimes by σ which stands for the corresponding allocation state) and identified by the arcs which it cuts. Thus $\gamma_0 = \alpha_1^1\ \alpha_1^2 \cdots \alpha_1^m$ is the initial slice which intersects all initial arcs of chains. The terminal slice γ_T of a d-graph is similarly defined. As the processes progress the "current"

slice wiggles down the d-graph from the initial slice to γ_T as processes proceed to a finish. As both slices and chains are sets, the notation $\gamma \cap X_j$ will be used to designate the arc on chain X_j which is a component of γ. The "intersection" is thus literal too.

Readers who are familiar with Holt's occurrence-graphs [2] and Petri nets [3] will notice the similarity of the representation scheme above to those representations. The similarity is not accidental. The present representation was suggested by Holt's occurrence graphs which in turn are very closely related to Petri nets.

For any slice, γ, there are nodes (transitions) which lie above it (assuming the arrows on the arcs to point downwards) which constitute the predecessor set $P(\gamma)$ and nodes which lie below it which constitute the successor set $S(\gamma)$. Every node in the graph must belong to exactly one of these two sets, i.e. a slice partitions the nodes in the graph. It follows from the definition of the graph that $P(\gamma_0) = \phi$ and $S(\gamma_T) = \phi$ where γ_0 and γ_T represent the initial and terminal slices of the d-graph. An anti-symmetric relation E (earlier than) can be defined on two slices γ_1, γ_2 as follows: γ_1 E γ_2 if $P(\gamma_2) \supset P(\gamma_1)$ and $S(\gamma_1) \supset S(\gamma_2)$. The relation L (later than) is the converse of E. Thus γ_1 L γ_2 if $P(\gamma_1) \supset P(\gamma_2)$ and $S(\gamma_2) \supset S(\gamma_1)$, i.e. γ_1 E $\gamma_2 \Rightarrow \gamma_2$ L γ_1. Clearly γ_0 E γ_T. In fact γ_0 E γ and γ_T L γ for any slice γ distinct from γ_0 and γ_T. It should be noted that the inclusion in the definition of the two relations is strict. An immediate successor of a slice γ is a slice γ' such that γ' L γ and $E(\gamma')$ differs from $E(\gamma)$ by 1 node.

The complementary pair of relations E and L define a partial ordering on the slices γ of the graph. Thus in figure 3 neither of $\gamma_1 \; E \; \gamma_2$ and $\gamma_2 \; E \; \gamma_1$ are true since $S(\gamma_1) \not\supset S(\gamma_2)$ and $S(\gamma_2) \not\supset S(\gamma_1)$. Thus γ_1 and γ_2 are not ordered by the relation E (the same is true for L.)

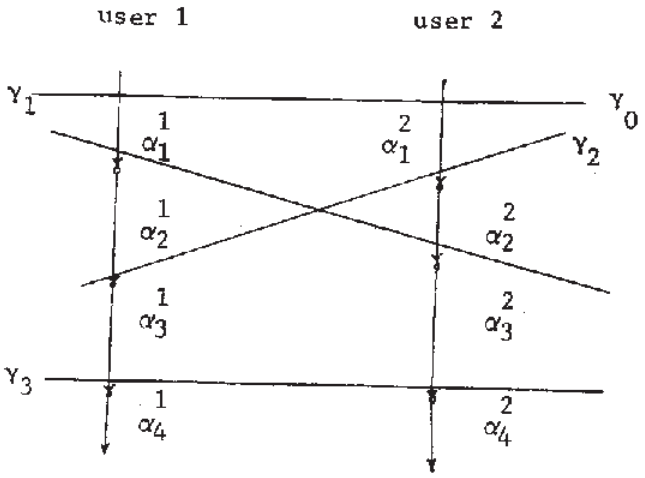

Figure 3

It can be shown that the slices of a demand graph form a distributive lattice under the relation E with γ_0 as the greatest element and γ_T as the least. Figure 4 illustrates the lattice of slices of the demand graph in figure 3. It will be noted that all the immediate successor slices of a slice γ lie one rank below γ and are connected to it by links. The least upper bound (l.u.b.) of two slices γ_1 and γ_2 is the slice γ $\gamma \; E \; \gamma_1 \wedge \gamma \; E \; \gamma_2$ and $\nexists$ a slice $\gamma' \ni \gamma' \; E \; \gamma_1$, $\gamma' \; E \; \gamma_2$ and $\gamma \; E \; \gamma'$.

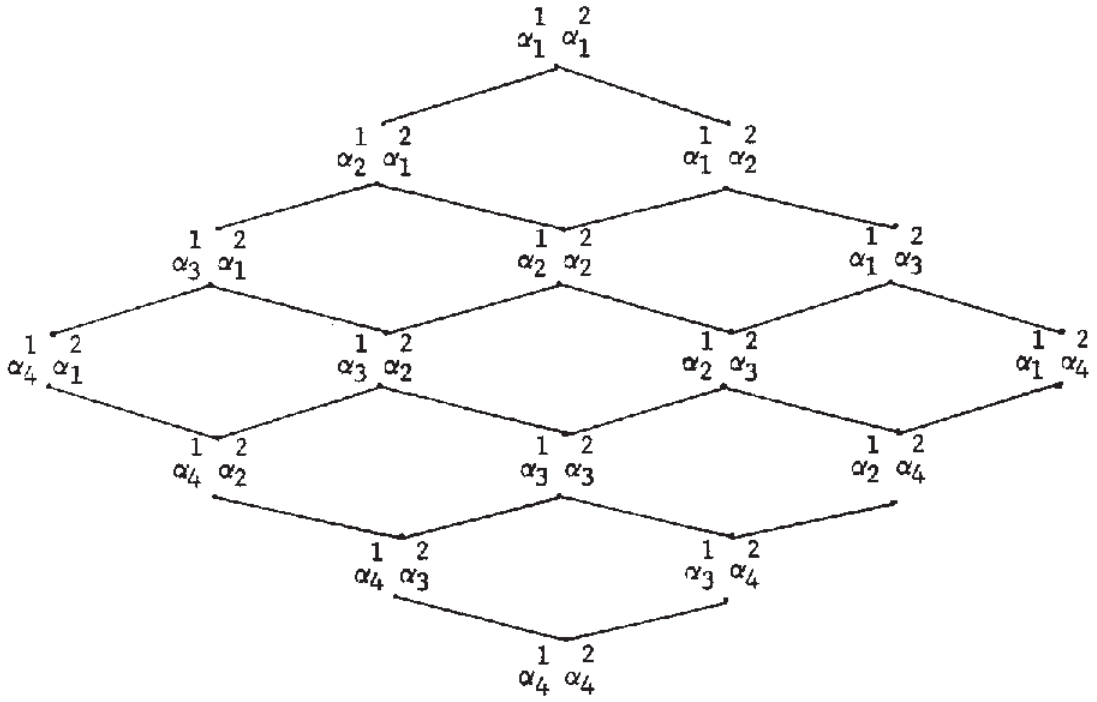

Figure 4

Thus in figure 3 the ℓ.u.b. of γ_1 and γ_2 is γ_0. The greatest lower bound (g.ℓ.b.) is similarly defined as: g.ℓ.b $(\gamma_1, \gamma_2) = \gamma$ iff $\gamma \mathrel{L} \gamma_1 \wedge \gamma \mathrel{L} \gamma_2$ and $\nexists$ a slice $\gamma' \ni \gamma' \mathrel{L} \gamma_1$, $\gamma' \mathrel{L} \gamma_2$ and $\gamma \mathrel{L} \gamma''$. Thus the g.$\ell$.b. of γ_1 and γ_2 in figure 4 is the slice $\overset{1}{\alpha_2}\ \overset{2}{\alpha_2}$.

In terms of the progress of the processes, again, the "current" slice starts out at the greatest element of the lattice of slices and moves down the lattice one link at a time to the least element of the lattice, the actual path taken depending on the rates of progress of the processes.

Several properties of the lattice of slices of a demand graph are interesting. Firstly it will be noted that in the lattice of figure 4 the sum of the indices of every node in a rank is the same. This is because nodes in a rank represent slices resulting from the same number of moves but distributed in all possible ways over the chains (where a "move" consists in passing over a transition). The number of moves is precisely the rank index (measured starting from the top-most node with 0). Thus the elements of rank 2 in figure 4 above are $\alpha_3^1\ \alpha_1^2$, $\alpha_2^1\ \alpha_2^2$ and $\alpha_1^1\ \alpha_3^2$ which slices are exactly two moves from $\alpha_1^1\ \alpha_1^2$. This property results in the following expression (due to Prof. C. L. Liu) for the number of elements in a rank:

no. of elements in rank n = coefficient of x^n in

$$(1+x+x^2+\ldots x^{n_1-1})\ (1+x+x^2+\ldots x^{n_2-1})\ (1+x+\ldots x^{n_3-1})\ldots\ldots(1+x+x^2+\ldots x^{n_m-1})$$

where n_i has the meaning assigned above, i.e. the number of arcs on the i^{th} chain. Thus there is one node at rank 0 and one at rank $L = (n_1 + n_2 + \ldots n_m)-m$ (the bottom-most node) and m nodes at ranks 2 and L-1 (unless some $n_i = 1$). In fact the lattice is symmetric i.e. the number of nodes at rank r is the same as that at rank L-r, $0 \leq r \leq L$. The other property of a distributive lattice is that the lengths of all paths between two elements of a lattice are the same and equal to the difference of the rank indices of the two elements.

The following definitions which will be used throughout the text can now be stated. A slice (of the demand graph) or node (of the lattice of slices of a demand graph) and the associated allocation state is said to be <u>feasible</u> if the sum of the demands associated with the arcs in the slice does not exceed the resource capacity of the system, i.e.

$$\sum_{i=1}^{m} d(\alpha^{i}_{r_i}) \leq C \qquad 1 \leq i \leq m, \quad 1 \leq r_i \leq n_i$$

Thus a feasible slice represents a potential allocation state of the multi-user system. An infeasible slice corresponds to an unattainable allocation state. A feasible slice or node and its associated allocation state is said to be <u>safe</u> if there exists a path in the lattice, passing only through feasible slices, from it to the terminal slice, i.e. if there exists a sequence of moves leading from the slice to the terminal slice by means only of feasible slices (see figure 5). Interpreted in terms of the processes, a safe slice represents an allocation state with the property that the processes can complete without deadlock. It should be noted that if the system ever attains an unsafe state then deadlock must occur eventually but a safe allocation state does not imply that no fear of deadlock exists. This is stated in theorem 0 below.

<u>Theorem</u> 0. In a multiprocess system with demand graphs of the type described, processes in a partial state of completion can proceed to thorough completion in a finite time iff the corresponding allocation state, σ, is safe.

By "thorough" completion is meant completion without omission of phases. Every phase is assume to last for a finite time.

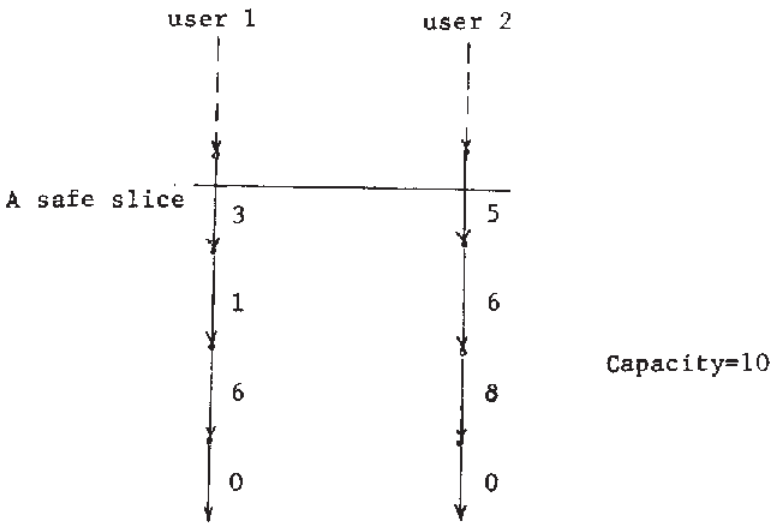

Figure 5

Proof: The following easily proved lemma is useful in the proof.

Lemma: Every path from a node γ in the lattice of slices to the terminal slice (γ_T) contains at least one node that contains any given arc $\alpha^i_{r_i}$ of a chain i that lies between the arc $\gamma \cap X_i$ and the arc $\alpha^i_{n_i}$ (inclusive), where $\gamma \cap X_i$ is the arc on chain i that is contained in γ.

Proof: The length of any path from γ to γ_T is $\ell = L-r$ where r is the rank of γ. Thus $\ell =$ number of transitions in $S(\gamma)$.

Now $\alpha^i_{r_i}$ is either bounded by two transitions below γ or belongs to γ or γ_T. If it belongs to γ and γ_T then that is the node which contains $\alpha^i_{r_i}$ and the result follows. (It could of course belong to both γ and γ_T.)

If $\alpha^i_{r_i}$ is not in γ or γ_T, let t_1 represent the transition at the tail (i.e. top) of arc $\alpha^i_{r_i}$. Then the path from γ to γ_T must have involved a move which involved crossing t_1, for all the nodes on the path have to have distinct labels and there are $\ell+1$ such nodes so that ℓ transitions must have been crossed which is exactly the number of transitions in $S(\gamma)$. Thus the slice following this move contains $\alpha^i_{r_i}$. QED.

Proof of Theorem 0 continued:

Let σ be safe. Then $\exists$ a path π consisting only of feasible slices from γ to γ_T where γ corresponds to σ and γ_T to σ_T, the terminal allocation state. Each of these slices represents an allocation state which is attainable and so does σ. Thus by letting the processes continue but granting only those requests which lead to the allocation state corresponding to the next slice

on the path π one ensures that the terminal allocation state will be reached in a finite amount of time (since each phase of a process is assumed to last only a finite time). Moreover, by virtue of the lemma above every phase of every process is gone through. Thus all the processes can reach thorough completion in a finite time.

Conversely, if σ is unsafe then thorough completion cannot be reached in a finite time. For every path between γ and γ_T has at least one infeasible slice on it as a consequence of the unsafeness of γ (it is assumed that since σ corresponds to an allocation state already attained, γ is feasible). Now if all the processes actually went to thorough completion then they must have done so in ℓ states in addition to σ and σ_T and $\ell-1 = S(\gamma)$ is the number of transitions which must all be crossed for thorough completion. The slices corresponding to the ℓ states all appear in the lattice (since the lattice displays all slices) and must moreover lie on a connected path from γ to γ_T since they can be reached from γ and since γ_T can be reached from them. Thus this path is a path from γ to γ_T consisting of only feasible slices (since the states were actually attained) which means σ is safe -- a contradiction. Thus at some point the system must have reached a deadlock i.e. a point where none of the unfinished processes could get the additional requests they sought. As no process can release any resources (if one could then that process could proceed which is impossible by the definition of deadlock) this situation must continue indefinitely. Thus the processes cannot go to thorough completion in a finite amount of time. QED.

Cor: The slice resulting from any number of moves from an unsafe slice is also unsafe.

Proof: The result follows from the fact that all the paths from a successor of γ are contained in the set of paths from γ. Thus even if a move results in the return of some resources the resulting state is still unsafe. It is important, therefore, that the system be kept from entering an unsafe state. Although the theorem is a simple consequence of the definition, it is central to the entire discussion to follow, for now deadlock-freeness is equivalent to safeness of the allocation state.

The situation where the only a priori knowledge of demand behaviour is the maximum amount that will be needed by each process (max_i for process i) can be represented in two ways which are shown in figures 6 and 7. In figure 7 the i^{th} chain has $max_i + 2$ arcs.

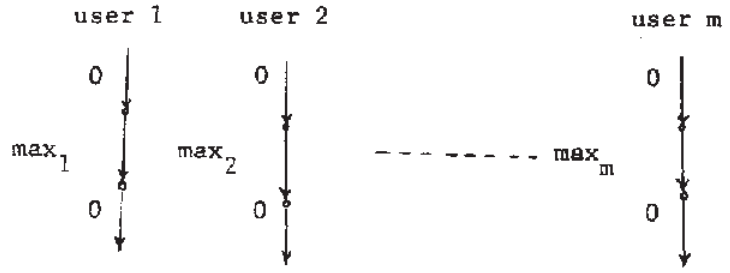

Figure 6

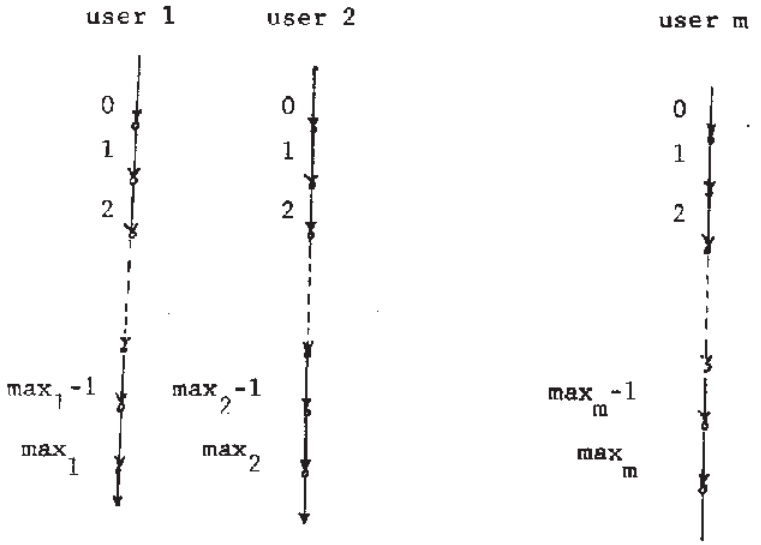

Figure 7

Remark: In either model it is clear that the existence of a path in the lattice of slices from the current slice, γ, to γ_T has as a neccessary and sufficient condition the existence of an ordering $i_1\ i_2 \ldots i_m$ of the users such that the slice consisting of γ with $\gamma \cap X_{i_1}$ replaced by the arc with $\max_{i_1}$ (i.e. $\alpha^1_{n_i-1}$) be feasible, γ with $\gamma \cap X_{i_1}$ replaced by $\alpha^i_{n_{i_1}}$ and $\gamma \cap X_{i_2}$ by $\alpha^i_{n_{i_2}-1}$ be feasible and so on which is exactly theorem 1 of section 2.3 of Habermann's thesis [1]. However the representation of figure 7 is to be preferred as it allows for representation of the situation when only a part of the maximum demand has been allocated (as only that much is needed)--a situation which does actually occur with an allocation mechanism that tries to keep down unused resources. The representation scheme of figure 7 exactly models the situation Habermann studies if one allows some phases to take zero time (corresponding to a request for more than one unit of addition resource).

Tests for Safeness

The previous section showed that for the model chosen safeness of the allocation state is both necessary and sufficient for the absence of potential deadlock. While the definition of safeness provides a condition which can indeed be tested for, it appears as though it requires an exhaustive search of the lattice to find a sequence of feasible slices from a given slice to the bottom. Alternatively one could construct the lattice before any process is allocated any resources and mark off all the unsafe slices so that one need only check if a slice is marked or unmarked to determine its safeness or otherwise. The latter scheme has the principal disadvantage that addition of another process requires a complete recomputation of the safeness of slices which implies a large hiatus in the execution of running processes if the process is introduced while the existing processses are partially along. For this reason only "incremental" tests will be considered i.e. ones that only test the next slice for safeness. This has the additional advantage that only those slices that actually become current by virtue of requests are examined for safeness which could result in a saving in computation overhead. Moreover, it should be noted that none of the tests in the entire discussion to follow require construction of the lattice of slices. The lattice is only a tool for analysis and exposition. An algorithm is presented below which is a non-exhaustive procedure for constructing a feasible sequence from any slice to the bottom-most one (it becomes exhaustive only in the worst case).

Safeness Algorithm

Let γ be the slice whose safeness is to be examined.

1. Pick a chain X_i of the d-graph in some way (perhaps arbitrarily).

2. Construct a sequence of moves (which as described earlier consist of moving from an arc to the next one across a transition) down X_i so that the slice resulting from each move is feasible and the last resulting slice γ' has the property

$$d(\gamma' \cap X_i) \leq d(\gamma \cap X_i)$$

and the sequence is maximal i.e.

$$d(\text{i.s.}(\gamma' \cap X_i)) > d(\gamma' \cap X_i)$$

where i.s. is the immediate successor function. If such a sequence cannot be constructed go to step 3; if it can, replace γ by γ' and repeat the procedure from step 1 onwards until γ_T, the bottom-most slice, is reached.

3. Pick another unused chain X_j, $j \neq i$, and go to step 2 if one can be found.

It is obvious that even when the chain X_i in the first step is picked randomly from $\{X_1, X_2 \ldots X_m\}$, the algorithm can at worst degenerate into an exhaustive search of paths.

The power of this algorithm is revealed in the following theorem.

Theorem 1 An allocation state σ is safe iff a full (i.e. up to γ_T) sequence starting from the slice γ corresponding to σ can be constructed using the safeness algorithm above.

Proof. The "if" part is obvious since when a full sequence is found it satisfies the requirements for γ to be a safe slice i.e. for σ to be a safe state.

Now for the "only if" part. Suppose the algorithm fails to generate a full sequence and yet suppose σ is safe.

Then as σ is safe there exists a feasible sequence Σ of slices from γ to γ_T.

Moreover, as a consequence of the definition of d-graphs the last arc on every chain has the number (demand) 0 associated with it (as does the initial arc) while every other arc has a number greater than equal to 0 associated with it. As a result if the algorithm fails to generate a full sequence, it follows that there is a slice γ_1 and an arc, α_i, on each chain X_i such that

$$\forall i \quad [\gamma_1 - \gamma_1 \cap X_i]\cdot\alpha_i \quad \text{is an infeasible slice}$$
$$1 \leq i \leq m$$

where $[\gamma_1 - \gamma_1 \cap X_i]\cdot\alpha_i$ means the slice formed by using the same components as γ_1 except for the replacement of α_i for the arc $\gamma_1 \cap X_i$. This follows as the only way for the algorithm to fail to provide a full sequence is if at some stage there is no chain down which it can proceed without reaching an infeasible slice.

Let S be the set $\{\alpha_i \mid 1 \leq i \leq m\}$ and j be an integer $(1 \leq j \leq m)$ such that α_j is the first element of S to appear in a slice γ_0 of Σ, the sequence of feasible slices from γ to γ_T.

Then $\gamma_0 \cap X_i \leq \alpha_i \quad \forall i \quad \begin{matrix} 1 \leq i \leq m \\ i \neq j \end{matrix}$ where

"$\leq$" means earlier than or the same as".

Therefore, $d(\gamma_0 \cap X_i) \geq d(\gamma_1 \cap X_i) \; \forall i \quad \begin{matrix} 1 \leq i \leq m \\ i \neq j \end{matrix}$

This follows as the consequence of the algorithm which has the property that each intermediate slice γ', at which control goes back to step 1 of the algorithm, intersects every chain X_i at the arc with the smallest (demand) number between γ and γ'. Thus if the algorithm cannot proceed beyond γ_1 (step 2 getting blocked at α_i), then α_i has the largest number of any arc between $\gamma_1 \cap X_i$ and α_i i.e. $\gamma_1 \cap X_i$ has the smallest number on chain X_i between α_i' and α_i.

As γ_0 is feasible, replacing any of the terms in the corresponding inequality, $\sum_{k=1}^{m} d(\gamma_0 \cap X_k) \leq$ Capacity, by smallest terms cannot invalidate it. Thus

$$\sum_{\substack{k=1 \\ k\neq j}}^{m} d(\gamma_1 \cap X_k) + d(\gamma_0 \cap X_j) \leq C$$

i.e.

$$\sum_{\substack{k=1 \\ k\neq j}}^{m} d(\gamma_1 \cap X_k) + d(\alpha_j) \leq C$$

i.e. the slice $[\gamma_1 - \gamma_1 \cap X_j]^{\bullet}\alpha_j$ is feasible which is absurd as by assumption α_j had the property that $[\gamma_1 - \gamma_1 \cap X_j]^{\bullet}\alpha_j$ is infeasible.

However a full sequence such as Σ from γ' must contain at least one slice containing each α_i. Thus Σ cannot exist, i.e. σ is unsafe. QED.

Figure 8 illustrates how the algorithm constructs a (full) sequence

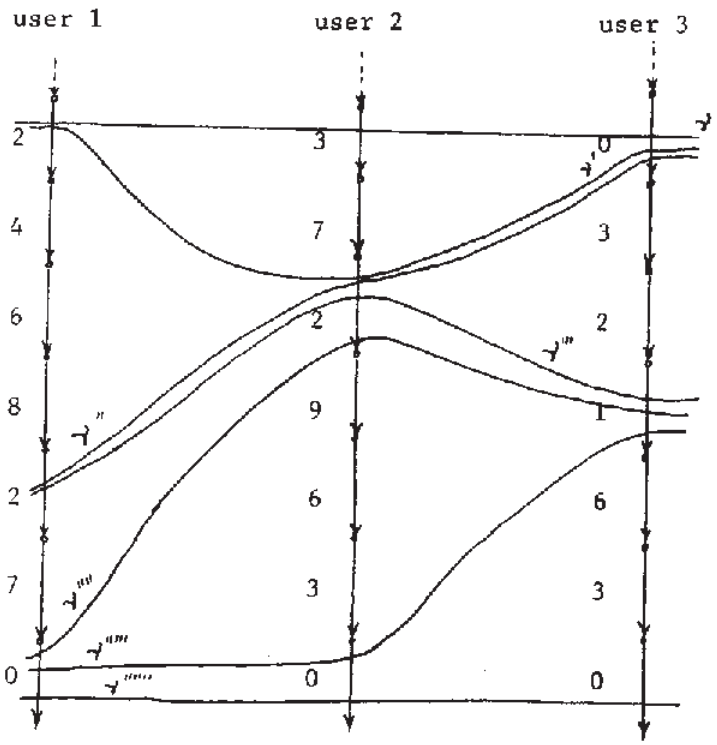

Figure 8

what the above theorem states can be reinterpreted as saying that no back-tracking is necessary (except to the extent that step 3 of the algorithm is used) in the construction of the sequence from γ to γ_T using the safeness algorithm. Thus if a <u>macro-move</u> (a set of moves of the type leading from γ to some $\gamma'''\cdots'$ satisfying the conditions of the safeness algorithm) is applicable at γ it may be applied fearlessly with the assurrance that the sequence is fully estensible if σ (the state corresponding to γ) is safe . This leads to corollary 1.

Cor. 1 If an allocation state σ is safe and a request for an additional allocation to user i is made with a consequent potential allocation state σ_i, then σ_i is safe if there exists a partial feasible sequence from the slice γ_i (corresponding to σ_i) to a slice γ_i' with the property that

$$d(\gamma_i' \cap X_i) \leq d(\gamma \cap X_i)$$

and $d(\gamma_i' \cap X_j) \leq d(\gamma \cap X_i)$ for $j \in \{j_1, j_2, \ldots, j_k^n \mid \text{integers in } (1,m)\}$ and $j \neq i$

[In particular $\gamma_i' \cap X_j = \alpha_{n_j}^j$ satisfies this condition]

and $\gamma_i' \cap X_j = \gamma_i \cap X_j = \gamma \cap X_j$ for $j = \{1,2,\ldots m\} - \{j_1, j_2, \ldots, j_k\}$ and $j \neq i$

This follows from the fact that the set of moves from γ to γ_i' (by way of γ_i) specified by the sequence is a macro-move in the construction of a sequence from γ to γ_T be means of the safeness algorithm, the existence of such a sequence being guaranteed by the safety of σ.

Cor. 2 If an allocation state σ is safe and an allocation request from a user i consists of a de-allocation (or returning of resources) leading to a potential allocation state σ_i, then σ_i is safe.

This follows from the fact that $\gamma - \gamma_i$ is the partial sequence fulfilling the conditions of Cor. 1 above.

The two corollaries above point out that the task of determining the safeness of an allocation state can sometimes be considerably simplified when its predecessor is known to be safe. In particular it is known that the initial slice of a d-graph is always safe if the demand associated with every arc is less than the capacity of the system (one-user-at-a-time completion is always possible) so that this fact can be used if resources are always allocated in a way that keeps the resulting allocation state safe.

As a further refinement it should be noted that in using the safeness algorithm to constrct a sequence from node γ for purposes of determination of the safeness of the allocation state σ (corresponding to γ) it is not needed to go as far as γ_T. By virtue of the single-chain character of the macro-moves the construction process can be stopped at slice γ' just before the last macro move as only one chain (X_j) remains then and the numbers on the arcs $\gamma' \cap X_i$ ($i \neq j$) are 0 while those on the remaining arcs of X_j do not exceed the capacity C so that completion of the sequence is assured.

Proposed Research

Sections 2 and 3 above presented a scheme for representing the situation wherein several independent users share a limited amount of an unpre-emptable resources asynchronously according to announced schedules. These sections also showed how deadlocks may be prevented and indicated a simple way to test for safeness of a slice. It is clear that an extension is needed to the multi-dimensional case, i.e. the case where the unpre-emptable resources are of several kinds -- in this case a complication is introduced by the fact that not every pair of (demand-) vectors can be ordered. Moreover the different users in a system are not always independent, in the context of the example introduced in section 1 the construction companies may also be involved in joint projects. It is therefore proposed to seek a representation suited to the study of deadlock and its prevention for the general case where several interdependent users share many kinds of unpre-emptable resources, where users may spawn asynchronous sub-processes and where additional users may enter the system. Results of the kind obtained in the theorems of sections 2 and 3 and bounds on the lengths of tests for safeness will be sought. A graphical model seems to have several advantages in this connection.

The proposed research will also deal with more complex usage of resources. For instance the needs of a particular phase of a process could perhaps be met by several combinations of resource types. Alternatively some of the resources may be versatile and be capable of providing more than one kind of service, perhaps even simultaneously. Or again the needs of a user may depend on the

way in which a decision based on other factors goes. Several related decisions of the latter kind may appear in the course of a process. A representation-scheme for these situations will be sought and its properties investigated in the context of the prevention of deadlocks. The concepts of "disjunctive inputs and outputs" from Estrin and Martin [5] and "conflicts" from Petri nets [3] provide an initial basis for this part of the proposed research.

The thesis research will also consider, to varying extents, the following areas in which interesting problems arise. The first one is that of cyclicity of a users needs. Cycles in demand graphs are difficult to handle because the slices do not form a lattice under the successor relation and because unlike in the acyclic case the safeness of a slice may depend on the parts of demand graphs above the slice in addition to those below it. The second area is that of asynchronous interactive systems where the necessity of providing a reasonably small response time may require pre-emption of resources even when the allocation scheme prevents the occurrence of deadlocks. Out-right pre-emption has been investigated by Coffman and Shoshani [6]. The pre-emption of interest here is of a recoverable roll-back kind, i.e., effectively moving back up a user's demand graph so as to release the needed amount of resources without loss of intermediate results of the computation. Interesting questions arise regarding the completion of the processes and a phenomenon similar to thrashing in page turning schemes. In this same context resources which cannot always be pre-empted, such as core memory allocation to some supervisory routines,

also present interesting problems. The third area is that of optimality of system capacity in the context of a given user mix and the trade-off between the number of safe states and the size of the pool.

References

[1] Habermann,A.N., On the Harmonious Cooperation of Abstract Machines, Ph.D. Dissertation,Technological University of Eindhoven,1967.

[2] Holt,A.W.,et al, Final Report of the Information System Theory Project, Applied Data Research Report Number 6606,February 1968.

[3] Petri,C.A.,Kommunikation mit Automaten,Schriften des Reinisch-West falischen Inst. Instrumentelle Math. and der Universitat Bonn,Nr. 2, Bonn,1962.

[4] Černikov,S.N., "Algebraic Theory of Linear Inequalities," American Mathematical Society Translations, Series 2, Number 69,1968,pp 147-203.

[5] Martin,David, and Gerald Estrin, "Models of Computations and Systems--Evaluation of Vertex Probabilities in Graph Models of Computations," JACM, Volume 14, Number 2, April 1967, pp 281-299.

[6] Shoshani,A. and E.G.Coffman, Detection Prevention and Recovery from Deadlocks in Multiprocess Multiple Resource Systems, Technical Report 80, Computer Science Laboratory, Department of Electrical Engineering, Princeton University, October 1969.