Abstract:

Application of a deadlock-avoidance strategy can be profitable if the number of resources actually in use is less than the number claimed. Most of the system overhead introduced by such a strategy is in the allocation safety test, because this test is applied every time a resource is requested. It is shown in this report that deadlock-avoidance strategies range from most conservative to most permissive with interesting alternative strategies in between. It seems that a strategy close to the conservative policy is adequate for many resource allocation systems. Such a policy has the advantage of a simple safety test which requires only a fixed (and very small) number of add-and-test operations.

A Spectrum of Deadlock-Avoidance Strategies

by

A.N. Habermann

TECHNICAL REPORT SERIES

Editor: Dr. B. Shaw

Number 53


© 1973 University of Newcastle upon Tyne.

Printed and published by the University of Newcastle upon Tyne, Computing Laboratory, Claremont Tower, Claremont Road, Newcastle upon Tyne, NE1 7RU.
HABERMANN, Arie Nicolaas

A spectrum of deadlock-avoidance strategies.

[By] A.N. Habermann


(University of Newcastle upon Tyne, Computing Laboratory Technical Report Series, no. 53).

12"

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE.

Suggested classmarks (primary classmark underlined)

Library of Congress:
Dewey (17th): 001.64404
U.D.C.: 681.322.06

Suggested keywords

MULTIPROGRAMMING
PROCESS INTERLOCK
RESOURCE ALLOCATION
SYSTEM DEADLOCK

Abstract

Application of a deadlock-avoidance strategy can be profitable if the number of resources actually in use is less than the number claimed. Most of the system overhead introduced by such a strategy is in the allocation safety test, because this test is applied every time a resource is requested. It is shown in this report that deadlock-avoidance strategies range from most conservative to most permissive with interesting alternative strategies in between. It seems that a strategy close to the conservative policy is adequate for many resource allocation systems. Such a policy has the advantage of a simple safety test which requires only a fixed (and very small) number of add-and-test operations.

About the Author

Professor Habermann, who is from Carnegie-Mellon University, Pittsburgh, is a Visiting Professor in the Computing Laboratory of the University of Newcastle upon Tyne between January and September, 1973.
Introduction

Designers of operating systems have taken quite different attitudes in the past with regard to detection and avoidance of system deadlocks arising from the allocation of non-preemptive resources. Opinions on the matter can be characterized by the following phrases:

1) "Deadlocks are not likely to occur, so don't bother"
2) "Restrict admission to the set of competitors so that deadlocks cannot occur"
3) "Test the allocation state every time it is about to change and allocate only if a deadlock cannot arise".

If it is true that deadlocks are not likely to occur, it may be that such happy circumstances have been assured by means of an abundant pool of resources. This situation can be maintained by purchasing more equipment when the average programming load increases or by artificially restricting the number of jobs at a low figure so as to make the occurrence of deadlocks unlikely. The drawback of either method is undesirable underutilization of equipment.

The second opinion about deadlocks usually result in ad-hoc methods that have to do with the order in which resources can be requested, or that restrict the number of jobs competing for a certain set of resources. The advantage obviously is that deadlocks cannot occur, but implementations of this sort may suffer from the same drawback as those based on the first opinion.

The third opinion is expected to lead to the optimal utilization of resources. Current implementations based on this idea, however, have the serious drawback of significant overhead in the form of a dynamic safety test that is executed each time a resource is allocated or released. This disadvantage is usually considered as the justification for taking the first or second point of view. For, if one does not bother about deadlock detection, such overhead is totally absent. If the order of resource requests is tested or a restriction is imposed on the number of competitors, the necessary checks can frequently be implemented as compile-time or load-time tests. Even if such tests were carried out
dynamically, the runtime overhead is in all likelihood significantly less than with an allocation-safety test, because such tests are performed when a process requests admission to a group of competitors and that probably happens less frequently than requesting allocation or releasing a resource.

The purpose of this paper is to show that it is possible to reduce an allocation-safety test to so little that objections against the overhead involved are no longer valid, and in such a way that not much of the accuracy of a full-fledged safety test is lost. It will be shown that the overhead can be reduced significantly if two kinds of situations are explored:

a) situations of which we can predict that a resource cannot safely be allocated

b) situations of which we are sure that allocation to certain processes will definitely not cause a deadlock in the future.

If we can recognize situations as mentioned under a), this knowledge can be exploited to prevent the occurrence of such situations. This then means fewer executions of the allocation-safety test, in particular in cases for which the test would come to a negative conclusion anyway.

We will see that knowledge of situations as mentioned under b) can be used to reduce considerably the number of separate tests that constitute the allocation-safety test. It has already been shown \([1,2,3]\) that a linear algorithm will suffice in the special case of many resources of one kind. The ideal is to reduce the allocation-safety test to a small and fixed number of tests independent of the number of resources or processes involved. We cannot get below a minimum amount of work which must be done irrespective of the chosen policy, viz. the check for available resources and request-overflow (the latter if a process stated its needs in advance). We will see that a simple safety test indeed suffices in many situations.

In section 2 we discuss the quantities that constitute a resource allocation state. Allocation to a process is expressed in terms of a promotion of this process to a higher rank. The notion of promotion is
used in section 3 to describe a safety test that is both necessary and sufficient to avoid the occurrence of a deadlock in the resource allocation. In section 4 we discuss the correlation between safe states and the number of processes with their specific demands.

A range of strategies varying from most conservative to most permissive is presented in section 5. This is followed by a discussion of the relation between utilization and the strategies of the range introduced in section 5. We will find in section 6 that a policy close to the conservative strategy is adequate for many resources allocation systems. This result is the primary contribution of this report. In the final section we discuss briefly how one of the simple strategies could be implemented. In the conclusion we give some thought to further investigations that may lead to useful extensions of this work.
2. Allocation state representations.

We confine ourselves in this report to the special case of one single type of resource. A set $S$ consists of processes $C_1, \ldots, C_n$ that are contending for resources $R_1, \ldots, R_{tot}$, where $n \geq 1$ and $tot \geq 2$. Associated with a process $C_i$ are two variables $claim_i$ and $rank_i$ that have a value in the range $0$ through $tot$. The quantity $claim_i$ is constant during the time a process is in set $S$ and it indicates that process $C_i$ will not attempt to acquire more resources than indicated by $claim_i$. The quantity $rank_i$ changes with allocations to or releases by process $C_i$; it indicates the number of resources that can still legitimately be requested by process $C_i$. Thus if we introduce a quantity $alloc_i$, being the number of resources allocated to process $C_i$, we find that the relation

$$rank_i = claim_i - alloc_i$$

is always true.

An allocation state is determined by the number $tot$, the total number of resources, and the numbers $rank_i$ and $claim_i$. We sometimes use for convenience the quantity

$$rem = tot - \sum alloc_i$$

which indicates how many resource there are left at a particular instant. A "realizable" allocation state satisfies

a) $0 \leq rank_i \leq claim_i \leq tot$

b) $rem \geq 0$

We assume that all allocation states that we consider are realizable states.

Examples

E1. $tot = 7$  
$claim = (2 2 4 4 5)$  
$rem = 0$  
$rank = (1 1 2 2 4)$ hence $alloc = (1 1 2 2 1)$
E2. \[ \text{tot} = 7 \]
\[ \text{claim} = (2 \ 2 \ 4 \ 4 \ 5) \quad \text{hence} \quad \text{rem} = 0 \]
\[ \text{rank} = (0 \ 0 \ 3 \ 3 \ 4) \quad \text{alloc} = (2 \ 2 \ 1 \ 1 \ 1) \]

E1 is an example of a so-called "unsafe" state, because all processes involved may ask for more resources and, if they indeed do, allocation runs into a deadlock. E2 is a "safe" state, because the processes with claim = 2 will not ask for any more resources and they will eventually return enough resources to satisfy any legitimate request by the others.

In a report preceding this one, various criteria have been discussed by means of which it can be decided whether or not a given allocation state is safe with respect to deadlocks [3]. In another paper it has been shown that a deadlock cannot occur if the only state transitions which are permitted transform a safe state into another safe state [4].

We will use here a method of testing for safe states as developed in the preceding report referred to above. This method is based on the notion of "promotions".

Given a realizable allocation state (i.e. given a set of numbers \text{tot}, \text{rank}_i and \text{claim}_i that satisfy relations a) and b)),

Let \[ \text{category}_k = \{ C_i \mid \text{claim}_i = k \} \]

Let \[ \text{class}_k = \{ C_i \mid \text{rank}_i = k \} \quad \text{for} \quad 0 \leq k \leq \text{tot}. \]

A process \( C_i \) belongs to exactly one category and also, at any instant, to exactly one class. The category structure is independent of allocations, the class structure, however, changes with allocations and releases. When a resource is allocated to a process \( C_i \) in \( \text{class}_k \), its rank is decremented by one, so this process \( C_i \) leaves \( \text{class}_k \) and enters \( \text{class}_{k-1} \). A process has all the resources it ever needs when it reaches \( \text{class}_0 \).

The number of elements in a \( \text{category}_k \) is denoted by \( n_k \) and the vector of all these numbers is denoted by

\[ \Pi = (n_1 \ n_2 \ \ldots \ n_{\text{tot}}) \]

(Since no process claims zero resources, \( \text{category}_0 \) is void, so we can omit \( n_0 = 0 \)).
With a class\(_k\) we associate a number denoted by \(p_k\) which we call the number of "promotions" that have taken place from class\(_k\) to class\(_{k-1}\). The value of a particular number \(p_k\) can be derived from the given values for claims and ranks with the following rule: a process \(C_i\) adds one to \(p_k\) if and only if \(\text{rank}_i < k \leq \text{claim}_i\).

Intuitively, a process \(C_i\) contributes one to \(p_k\) if it "passed through" class\(_k\), that is to say, if it started in a class equal to or below class\(_k\) and has arrived at a class above class\(_k\) (i.e. at a class in the range class\(_0\) through class\(_{k-1}\)). Observe, however, that the numbers \(p_k\) do not depend on the history of allocation and releases; the values are derived solely from the given claims and ranks.

The vector of all promotions is denoted by

\[
\mathbf{p} = (p_1, p_2, \ldots, p_{\text{tot}}).
\]

(Note that \(p_0\) is void, because no process is ever promoted beyond class\(_0\) and so we can omit \(p_0 = 0\)).

Examples.

E1: \(\text{tot} = 7\)

\[
\begin{align*}
\text{claim} &= (2 2 4 4 5) \quad \text{hence} \quad \mathbf{n} = (0 2 0 2 1 0 0) \\
\text{rank} &= (1 1 2 2 4) \quad \mathbf{p} = (0 2 2 2 1 0 0)
\end{align*}
\]

E2: \(\text{tot} = 7\)

\[
\begin{align*}
\text{claim} &= (2 2 4 4 5) \quad \text{hence} \quad \mathbf{n} = (0 2 0 2 1 0 0) \\
\text{rank} &= (0 0 3 3 4) \quad \mathbf{p} = (2 2 0 2 1 0 0)
\end{align*}
\]

In E1 element \(p_3 = 2\), because the processes in category\(_3\) have reached class\(_3\), so these processes passed through class\(_3\). In E2 element \(p_3 = 0\) but this time \(p_1 = 2\), because the process in category\(_5\) reached class\(_0\).

Observe that the values of claim and rank cannot conversely be derived from given values of \(\mathbf{n}\) and \(\mathbf{p}\).

Example.

E3: \(\mathbf{n} = (0 1 1 0)\)

\[
\mathbf{p} = (1 2 1 0)
\]

8.
The fact that $p_2 = 2$ means that both processes have been promoted from class 3 to class 1, i.e. the processes reached at least class 1. Since $p_1 = 1$, one process has reached class 0 and the other is still in class 1. There is, however, no way of telling whether the process of category 3 or the one of category 2 was promoted to class 0. It is therefore not possible to determine uniquely the ranks of all the processes.

We conclude that we apparently lose some information by going from claims and ranks to the vectors $n$ and $p$. We will see, however, that this reduction of information is no real loss with respect to avoiding deadlocks and we will also see that it enables us to simplify the safety test considerably.
3. The Safety-test expressed in promotions

Another way of visualizing promotions is described below. Associate with each class $\xi$ a fixed vector $\underline{u}_\xi$ consisting of $k$ ones followed by $(\text{tot}-k)$ zeros. We add $\underline{u}_\xi$ to a tot-dimensional vector denoted by $\underline{P}$ (initial value $0$) on account of process $C_i$ if and only if $\text{rank}_i < k \leq \text{claim}_i$. Thus, if $\text{rank}_1 = 2$ and $\text{claim}_1 = 4$, the vectors $\underline{u}_4$ and $\underline{u}_3$ are added to $\underline{P}$ on behalf of process $C_1$. The value of $\underline{P}$ represents an accumulative recording of the promotions.

Example

E4: $\begin{pmatrix} \underline{u}_1 \\ \underline{u}_2 \\ \underline{u}_3 \\ \underline{u}_4 \\ \underline{P} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 3 \\ 3 \\ 1 \\ 0 \end{pmatrix}$ The given value of $\underline{P}$ reflects one promotion from class $\xi_3$ and two promotions from class $\xi_2$, because $\underline{P} = 2 \underline{u}_4 + \underline{u}_3$

There is, of course, a direct relation between $\underline{P}$ and the promotion vector $\underline{p}$. A given value of $\underline{P}$ is uniquely decomposable into a sum of vectors $\underline{u}_k$ and the coefficients are precisely the elements of the promotion vector $\underline{p}$. This is expressed in the following lemma.

Lemma 1. Let $U$ be the tot*tot matrix of which the $k^{th}$ row is the vector $\underline{u}_k$; the relation between the promotion vector $\underline{P}$ and the vector $\underline{P}$ in which the totality of the promotions is recorded in the manner described above is:

$$\underline{P} = \underline{P} \cdot U$$

proof. All promotions from class $\xi_k$ through class $\xi_{\text{tot}}$ add one to element $P_k$, because all vectors $\underline{u}_i$, where $k \leq i \leq \text{tot}$, have a one in the $k^{th}$ position. None of the promotions from class $\xi_0$ through class $\xi_{k-1}$ contribute to element $P_k$, because all vectors $\underline{u}_i$, where $0 \leq i < k$, have a zero in the $k^{th}$ position.

Thus, $P_k = \sum_{i=k}^{\text{tot}} P_i$.

(1)
The $k$th column of matrix $U$ starts with $(k-1)$ zeros followed by all ones. So,

$$[P * U]_k = 0.P_1 + \ldots + 0.P_{k-1} + 1.P_k + \ldots + 1.P_{\text{tot}} = \sum_{i=k}^{\text{tot}} P_i$$  \hspace{1cm} (2)

The lemma follows from equations (1) and (2).

In the preceding report [3] it has been proved that the relation

$$P \leq \text{tot},$$

where $\text{tot} = (\text{tot} \ \text{tot}-1 \ \ldots \ 1)$, represents a necessary and sufficient condition for the safety of a realizable state. This result was obtained by showing that this safety condition is equivalent to other well-known safety criteria expressed in terms of claims and allocation. Some of the other ways in which this safety condition can be phrased are:

$$P_1 < \text{tot}, \ P_2 < \text{tot}-1, \ldots, \ P_{\text{tot}} < 1$$

or equivalently

for all $k \in \{1, \ldots, \text{tot}\}$ $P_k \leq \text{tot} + 1 - k$ or

for all $k \in \{1, \ldots, \text{tot}\}$ $\sum_{i=k}^{\text{tot}} P_i \leq \text{tot} + 1 - k$ or equivalently

$$P_1 + P_2 + \ldots + P_{\text{tot}} \leq \text{tot}$$

$$P_2 + \ldots + P_{\text{tot}} \leq \text{tot} - 1$$

$$P_3 \ldots + P_{\text{tot}} \leq \text{tot} - 2$$

$$\vdots$$

$$P_{\text{tot}} \leq 1$$

We can easily see that a safety test of linear complexity can be programmed which updates and inspects an array $P$ representing vector $P$ or, what amounts to the same thing, the partial sums of the promotions. Comparison with the algorithms given by Holt [1] and Russell [2] shows that those which are based on claims and allocation need somewhat less updating than an algorithm which uses promotions. However, all of these algorithms are probably too detailed as will be shown in the sequel.

Some useful properties of the matrix $U$ are expressed in lemmas 2, 3, 4 and 5.
Lemma 2. Let $\mathbf{a}$ be a tot-dimensional vector and let
\[ \mathbf{A} = \mathbf{a} \times \mathbf{U}; \] the elements of $\mathbf{a}$ and $\mathbf{A}$ are related by
the equations:
\[ A_k - A_{k+1} = a_k \quad \text{for} \quad 1 \leq k \leq \text{tot} - 1 \]
and
\[ A_{\text{tot}} = a_{\text{tot}} \]
proof. $A_k = \sum_{i=k}^{\text{tot}} a_i$, because the $k^{th}$ column of matrix $\mathbf{U}$
consists of $(k-1)$ zeros followed by all ones. The lemma
follows immediately after substitution.

Lemma 2 can be used to show that vector $\mathbf{P}$ is uniquely
decomposable as a sum of vectors $\mathbf{u}_k$. Suppose
\[ \mathbf{P} \times \mathbf{U} = \mathbf{P} = \mathbf{q} \times \mathbf{U} \]
Let $\mathbf{a} = \mathbf{P} - \mathbf{q}$, so $\mathbf{A} = \mathbf{a} \times \mathbf{U} = \mathbf{0}$. According to lemma 2, we have
\[ a_k = A_k = A_{k+1} = 0 \quad \text{for} \quad 1 \leq k \leq \text{tot} - 1 \quad \text{and} \quad a_{\text{tot}} = A_{\text{tot}} = 0 \]
Hence, $\mathbf{P} = \mathbf{q}$.

Lemma 3. Let $\mathbf{a}$ and $\mathbf{b}$ be $n$-dimensional vectors, let $\mathbf{M}$ be a matrix
with $n$ rows of non-negative elements; let $\mathbf{A} = \mathbf{a} \times \mathbf{M}$ and
\[ \mathbf{B} = \mathbf{b} \times \mathbf{M}. \]

\[ \mathbf{a} \leq \mathbf{b} \text{ implies } \mathbf{A} \leq \mathbf{B} \]
proof. $\mathbf{a} \leq \mathbf{b}$ means for all $i \in \{1, \ldots, \text{tot}\}$ $a_i \leq b_i$ and, if $c \geq 0$,
\[ a_i \times c \leq b_i \times c. \] Thus, $a_i \times m_{ij} \leq b_i \times m_{ij}$ for all relevant $j$,
because $m_{ij} \geq 0$. Hence, $A_j = \sum_{i=1}^{n} a_i \times m_{ij} \leq \sum_{i=1}^{n} b_i \times m_{ij} = B_j$
for all relevant $j$, or $\mathbf{A} \leq \mathbf{B}$.

Lemma 3 applies to matrix $\mathbf{U}$, because $\mathbf{U}$ has non-negative elements. The
converse of lemma 3 is not true as we can see in the following example.

E5: \[ \mathbf{a} = (2 \ 1 \ 0 \ 0), \text{ so } \mathbf{A} = (3 \ 1 \ 0 \ 0) \]
\[ \mathbf{b} = (1 \ 1 \ 1 \ 1), \text{ so } \mathbf{B} = (4 \ 3 \ 2 \ 1) \]

$\mathbf{a} \leq \mathbf{b}$ is not true, but $\mathbf{A} \leq \mathbf{B}$ is true.

(Note that multiplication of a vector $\mathbf{b}$, consisting of all ones, with
matrix $\mathbf{U}$ yields vector $\text{tot} = (\text{tot} \ \text{tot} \ ... \ 1)$. The safety condition
could therefore also be stated as $(\mathbf{b} - \mathbf{P}) \times \mathbf{U} \geq 0.$)
Lemma 4. Let $a$ be a tot-dimensional vector and let $A = a \ast U$;

$a \geq 0$ and $A_k = 0$ implies $A_{k+1} = 0$ for $k \in \{1, \ldots, \text{tot}-1\}$

proof. $a \geq 0$ implies $A = a \ast U \geq 0 \ast U = 0$ (lemma 3)

Thus, $A_k \geq 0$ for all $k \in \{1, \ldots, \text{tot}\}$ (1)

Suppose $A_i = 0$ where $1 \leq i < \text{tot}$.

$A_i - A_{i+1} = a_i$ (lemma 2), therefore $-A_{i+1} = a_i$ (2)

This relation, however, implies $A_{i+1} \leq 0$, since $a_i \geq 0$.

Combination of (1) and (2) yields $A_{i+1} = 0$.

Corollary. If $a \geq 0$ and there is an element $A_k = 0$, then all the following elements $A_{k+1}, A_{k+2}, \ldots, A_{\text{tot}}$ are also equal to zero.

Lemma 5. Let $a$ be a tot-dimensional vector and let $A = a \ast U$;

if $a \geq 0$, then $A_k = 0$ iff $a_i = 0$ for all $i \in \{k, \ldots, \text{tot}\}$

proof. If $A_k = 0$, then $A_{k+1}, \ldots, A_{\text{tot}} = 0$ (lemma 4), so

$a_i = A_i - A_{i+1} = 0$ (lemma 2).

If $a_i = 0$ for all $i \in \{k, \ldots, \text{tot}\}$, then $A_i = \sum_{j=1}^{\text{tot}} a_j = 0$ (see proof of lemma 2), in particular $A_k = 0$.

Corollary. If $a \geq 0$, $a_i = 0$ for all $i \in \{k, \ldots, \text{tot}\}$ is true if and only if $A_i = 0$ for all $i \in \{k, \ldots, \text{tot}\}$.
4. The basic relation enclosing $P$.

We assume that a process does not enter the set of competitors until the first resource it requests can be allocated to it. We assume furthermore that a process exits from the set of competitors as soon as it has returned all its resources. This means that a process, while in the set, holds at least one resource. It also means that a process whose claim = $k$ has at least been promoted away from class $k$, so all processes in category $k$ contribute at least to the number of promotions $P_k$ associated with class $k$. Recalling the definition of $n_k$ as being the number of processes in category $k$, we conclude that

$$n \leq P.$$  \hfill (1)

If we denote $n \times U$ by $N$ (analogous to $P \times U = P$), we find by lemma 3

$$n \times U = N \leq P$$  \hfill (2)

The claim of a process which contributes to a particular $P_k$ is greater than or equal to $k$ by virtue of the definition of the number $P_k$. This implies that no other process than those in category $k$ through category $k$ are able to contribute to $P_k$. Hence,

$$P_k \leq \sum_{i=k}^{k} n_i = N_k$$  \hfill (see proof of lemma 2)

and so

$$n \leq N$$  \hfill (3)

If we denote $N \times U$ by $\mathbb{N}$, we find by lemma 3

$$P \leq N \times U = N \times U$$  \hfill (4)

Combination of relations (1), (2), (3) and (4) gives us the basic relations on which all our further results rest:

$$n \leq P \leq N \leq P \leq \mathbb{N}$$

Recall that the safety condition requires

$$P \leq \text{tot},$$

where $\text{tot} = (\text{tot tot-1 \ldots 1})$

This leads to two observations:

(a) $N \leq \text{tot}$ is a necessary condition for safety, and
(b) $\mathbb{N} \leq \text{tot}$ is a sufficient condition for safety

14.
The significance of these two observations is in the fact that both can help us to reduce considerably the amount of work in the dynamic safety test ($P \leq \text{tot}$). Observation (a) can be exploited to spot situations which are not safe anyway, so we can prevent their occurrence, while observation (b) can be used to spot that part of the allocation state that forms no safety hazard anyway, so we can shorten the safety test accordingly.

The amount of work involved in testing $N \leq \text{tot}$ is the same as in the safety test $P \leq \text{tot}$: we must update and check $O(\text{tot})$ elements in either case. The test on $N$, however, has to be carried out only when a process enters or leaves the set of competitors, because the claim (from which $N$ is derived) does not change while a process participates in the competition.

The first significant conclusion is therefore:

a process that tries to join the set of competitors should be admitted only if the relation $N \leq \text{tot}$ remains true after admission of this process.

A discussion of programming an admission test follows later on, because it turns out that observation (b) has an impact on both safety test and admission test.

Suppose we required $N \leq \text{tot}$. This seems to be the best thing to do, because the safety test $P \leq \text{tot}$ and also the admission test $N \leq \text{tot}$ become entirely superfluous. We will shortly find, however, that such a strategy is likely to be unnecessarily restrictive in admitting processes to the set of contenders. We mark the strategy that requires $N \leq \text{tot}$ as the "most conservative" strategy.

The other extreme is not to require anything in addition to $N \leq \text{tot}$ and $P \leq \text{tot}$. This implies, of course, that both tests have to be fully applied at the appropriate moments. We mark this strategy as the "most permissive" strategy, because it includes only necessary rules and no (unnecessary) additional rules just for the sake of simplifying the tests. Note that the deadlock avoidance strategies as discussed in the literature [5] are even worse (computationally) than the most permissive strategy.
because none of those apply an admission test.

Example of the most conservative strategy for the case tot = 4.

E6: \( \text{tot} = (4 \ 3 \ 2 \ 1) \)

\[ \bar{n} = n \times U = n \times U^2 = n \times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}^2 = n \times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \\ 4 & 3 & 2 & 1 \end{pmatrix} \]

Thus, \( \bar{n} \leq \text{tot} \) is in this case equivalent to

\[ n_1 + 2n_2 + 3n_3 + 4n_4 \leq 4 \]
\[ n_2 + 2n_3 + 3n_4 \leq 3 \]
\[ n_3 + 2n_4 \leq 2 \]
\[ n_4 \leq 1 \]

We see that most conservative strategy restricts the set of competitors severely, e.g. it allows only one process whose claim is either 3 or 4 and it does not even tolerate an additional process whose claim = 2.

The table below shows a comparison between the conservative policy and the permissive strategy with or without admission test. It clearly demonstrates the advantage of applying an admission test.
<table>
<thead>
<tr>
<th># resources allocated to processes</th>
<th># cases</th>
<th># safe</th>
<th># not safe</th>
<th># caught in admission test</th>
<th># caught in safety test</th>
<th># accepted in conservative strat.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>--</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>3,1</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2,2</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2,1,1</td>
<td>30</td>
<td>15</td>
<td>15</td>
<td>9</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>1,1,1,1</td>
<td>35</td>
<td>15</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>--</td>
<td>--</td>
<td>2</td>
</tr>
<tr>
<td>2,1</td>
<td>12</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1,1,1</td>
<td>20</td>
<td>14</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>--</td>
<td>--</td>
<td>3</td>
</tr>
<tr>
<td>1,1</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>--</td>
<td>--</td>
<td>4</td>
</tr>
<tr>
<td>total</td>
<td>128</td>
<td>79</td>
<td>49</td>
<td>38</td>
<td>11</td>
<td>22</td>
</tr>
</tbody>
</table>

The conservative strategy allows only 22 out of 79 possible safe states to occur. The admission test catches most of the unsafe states (38 out of 49). The safety test is applied to 90 cases out of which 11 are not safe instead of to all 128 cases without an admission test finding 49 non-safe states. This means that adding the safety test reduced the probability of finding a non-safe state in the safety test approximately from 40% to 12%.

The essence of the conservative policy is that a process is not admitted unless all the resources it claims can safely be allocated to it right away. We can see that this is so in the following way. There are $n_k$ processes in category $k$ whose claim = $k$, so $k \times n_k$ resources are

17.
sufficient to satisfy those processes entirely. Therefore, all processes can simultaneously be satisfied if $\sum_{k} k \ast r_k \leq \text{tot}$.  

This sum, however, is exactly element $\eta_1$, because  

$$\eta_1 = \left[ N \times U \right]_1 = \sum_{i=1}^{\text{tot}} N_i = \sum_{i=1}^{\text{tot}} \sum_{j=1}^{\text{tot}} n_{ij} = \sum_{j=1}^{\text{tot}} \sum_{i=1}^{\text{tot}} n_{ij} = \sum_{j=1}^{\text{tot}} j \ast n_j.$$  

So, all the processes can simultaneously be satisfied if $\eta_1 \leq \text{tot}$, this is indeed required by the conservative policy, since $\eta \leq \text{tot}$ implies $\eta_1 \leq \text{tot}$.  

We will find in the following section that also $\eta_1 \leq \text{tot}$ implies $\eta \leq \text{tot}$, so the conservative strategy is precisely characterized by either relation: $\eta \leq \text{tot}$ or $\eta_1 \leq \text{tot}$.  

18.
5. A range of strategies.

We observed that \( \overline{\eta} \leq \text{tot} \) is a sufficient condition for safety, since the safety condition requires \( \overline{P} \leq \text{tot} \) and we found \( \overline{P} \leq \overline{\eta} \).

The conservative strategy which requires \( \overline{\eta} \leq \text{tot} \) is, however, not very attractive, because it allows so few safe states to occur. We try therefore to mitigate the stringent requirement of the conservative policy.

The relation \( \overline{\eta} \leq \text{tot} \) is equivalent to

\[
\overline{\eta}_k \leq \text{tot} + 1 - k \quad \text{for all } k \in \{1, \ldots, \text{tot}\}.
\]

Suppose the latter relation is true for some \( k \), though not for all \( k \) as in the case of the conservative policy. This is still useful information, because for such \( k \) the elements \( P_k \) don't need to be checked, since \( \overline{P}_k \leq \overline{\eta}_k \) for all \( k \).

At first sight we may fear that quite elaborate recording will be necessary in the course of allocating and releasing resources, so as to keep track of which elements \( P_k \) don't have to be checked. Fortunately, such fears are groundless, since it is only necessary to know the smallest \( k \) for which \( \overline{\eta}_k \leq \text{tot} + 1 - k \). It turns out, that if the relation holds for some \( k \), it automatically holds for all \( i \) in the range \( k \leq i \leq \text{tot} \). This is expressed in the following theorem.

Theorem. \( \overline{\eta}_k \leq \text{tot} + 1 - k \) for a given \( k \in \{1, \ldots, \text{tot}\} \)

implies \( \overline{\eta}_i \leq \text{tot} + 1 - i \) for all \( i \in \{k, \ldots, \text{tot}\} \)

proof. It is obvious that the theorem is true for \( k = \text{tot} \).

Let \( \overline{\eta}_k \leq \text{tot} + 1 - k \) be given where \( k \in \{1, \ldots, \text{tot} - 1\} \).

\[
\overline{\eta}_{k+1} = \overline{\eta}_k - N_k \quad \text{(lemma 2)}
\]

Suppose \( N_k = 0 \). Since \( \overline{n} \geq 0 \), \( N_i = 0 \) for \( k < i \leq \text{tot} \) (corollary of lemma 4).

Hence, \( \overline{\eta}_i \leq \text{tot} + 1 - i \) is certainly true, since \( 0 < \text{tot} + 1 - i \).

Suppose \( N_k \neq 0 \). Since \( \overline{n} \geq 0 \), we find by lemma 3 \( N_k \geq 1 \).

Thus,

\[
\overline{\eta}_{k+1} \leq \overline{\eta}_k - 1 \leq \text{tot} + 1 - k - 1 \leq \text{tot} + 1 - (k + 1)
\]

We can then prove step by step that \( \overline{\eta}_i \leq \text{tot} + 1 - i \) for all \( i \) in the range \( k \leq i \leq \text{tot} \).
Corollary. The vectors $P$ and $N$ have to be checked only up to the $k^{th}$ element if $\bar{\eta}_k \leq \text{tot} + 1 - k$, because $N \leq P \leq \bar{\eta}$, and $\bar{\eta}_i \leq \text{tot} + 1 - i$ for all $i$ in the range $k \leq i \leq \text{tot}$.

This result can be used to find a whole range of more suitable strategies in between the two extremes, the most conservative policy at one end and the most permissive strategy at the other end.

Let $S_k$ be the strategy which requires testing of the elements $P_i, \ldots, P_k$ in the safety test.

$S_0$ corresponds to the most conservative policy in which testing of $P$ is not at all necessary. $S_{\text{tot}}$ is the most permissive strategy, because it requires testing of the whole vector $P$.

A chosen strategy $S_k$ is enforced by limiting the admission of processes such that $\bar{\eta}_{k+1} \leq \text{tot} + 1 - (k + 1) = \text{tot} - k$ remains true at all times. Enforcement of strategy $S_k$ requires very little overhead, because $\bar{\eta}_{k+1}$ depends solely on claims, so $\eta_{k+1}$ changes only when a process is admitted or when a process leaves the set of competing processes. Moreover, all elements of $\bar{\eta}$ other than $\bar{\eta}_{k+1}$ are immaterial.

Expressed in terms of $n$ we have

$$\bar{\eta} = n \ast U^2 = n \ast \begin{pmatrix}
1 & 0 & 0 & \ldots & 0 \\
2 & 1 & 0 & \ldots & 0 \\
3 & 2 & 1 & \ldots & 0 \\
4 & 3 & 2 & \ldots & \ldots \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\text{tot} & \text{tot-1} & \text{tot-2} & \ldots & 1
\end{pmatrix}$$

The $(k+1)^{th}$ column of matrix $U^2$ starts off with $k$ zeros,

So $\eta_{k+1} = 1 \cdot n_{k+1} + 2 \cdot n_{k+2} + \ldots + (\text{tot} - k) \cdot n_{\text{tot}} = \sum_{i=k+1}^{\text{tot}} (i - k) \cdot n_i$.

Therefore, when a process whose claim $= j$ is admitted, $\eta_{k+1}$ does not have to be updated at all if $j \leq k$ and $(j - k)$ must be added if $j \geq k + 1$.

20.
Results of a comparison between the four strategies that apply to example E6 are listed below.

adm means admitted
rej means rejected

<table>
<thead>
<tr>
<th># resources</th>
<th># cases</th>
<th># safe</th>
<th># not safe</th>
<th>$S_0$ adm</th>
<th>$S_1$ rej</th>
<th>$S_2$ adm</th>
<th>$S_3$ rej</th>
<th>$S_4$ adm</th>
<th>$S_4$ rej</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>78</td>
<td>38</td>
<td>40</td>
<td>5</td>
<td>73</td>
<td>17</td>
<td>61</td>
<td>38</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>34</td>
<td>26</td>
<td>8</td>
<td>7</td>
<td>27</td>
<td>15</td>
<td>19</td>
<td>24</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>11</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>4</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>total</td>
<td>128</td>
<td>79</td>
<td>49</td>
<td>22</td>
<td>106</td>
<td>84</td>
<td>76</td>
<td>105</td>
<td>23</td>
</tr>
</tbody>
</table>

Policy $S_1$ seems to be a significant improvement over $S_0$.
Policy $S_4$ is apparently far too permissive. Strategy $S_2$ seems to be almost ideal. Be aware, however, that the figure of 52 rejections does not imply that strategy $S_2$ rejects all 49 unsafe states and by accident three of the safe states. The figure may have been formed by rejecting a few safe states and at the same time permitting some unsafe states to sneak in. There happen to be 46 unsafe states among the 52 rejected by $S_2$, so the chance of hitting an unsafe state in a test is only 3 in 76! Application of strategy $S_2$ would be a real improvement over $S_0$ or $S_4$ as we can see in the diagram. Even strategy $S_1$ is already a significant improvement over strategy $S_0$, while enforcement of $S_1$ does not require much more work than enforcement of $S_0$. Implementation of strategy $S_1$ is discussed in section 7 as an example of how simple it is to program a strategy close to the conservative one.

21.
6. Resource utilization in connection with the strategies.

The resource utilization is optimal if all the resources are in use. Allowing a total claim greater than the number of resources is therefore a useful thing to do if the processes do not use all their claimed resources all the time.

The total claim of a given set of competing processes equals \( \eta_i \), because the \( n_k \) processes in category \( k \) claim \( k \times n_k \) resources, so the total claim \( = \sum_{k=1}^{\text{tot}} k \times n_k = \eta_i \) (see end of section 4).

Define the "booking factor" \( b \) for a given set of competing processes by means of the equation

\[ \eta_i = b \times \text{tot} \]

We are particularly interested in those situations for which \( b > 1 \), in which case the value of \( b \) expresses that the total claim is that many times greater than the number of resources.

The total claim, and as a result the booking factor \( b \), cannot have an arbitrarily large value if an admission test is applied. The upper bound for \( b \) depends on the choice of strategy \( S_k \) and we therefore denote it by \( b_k \). Its value as a function of \( k \) is computed below.

The admission test of strategy \( S_k \) ensures that for \( 0 \leq k \leq \text{tot} \)

\[ \eta_{k+1} \leq \text{tot} - k \quad \text{(void in case } k = \text{tot)} \]

\[ N_i \leq \text{tot} + 1 - i \quad \text{for } 1 \leq i \leq k \]

(We know from the preceding section that \( \eta_{k+1} \leq \text{tot} - k \) implies \( N_j \leq \text{tot} + 1 - j \) for \( k + 1 \leq j \leq \text{tot} \), so these elements \( N_j \) do not have to be tested in the admission test).

\[ \eta_1 = \eta_2 + N_1 = \eta_3 + N_1 + N_2 = \ldots = \eta_{k+1} + \sum_{i=1}^{k} N_i \quad \text{(lemma 2)} \]

\[ S_0, \eta_1 \leq \text{tot} - k + \sum_{i=1}^{k} ( \text{tot} + 1 - i ) = ( \text{tot} - k ) + k ( \text{tot} + 1 ) - \frac{k}{2} k ( k + 1 ) \]

Thus, \( b \times \text{tot} = \eta_1 \leq (k + 1) (\text{tot} - \frac{k}{2} k) \),

and so the upperbound \( b_k = (k + 1) \left( 1 - \frac{k}{2 \times \text{tot}} \right) \) if strategy \( S_k \) is chosen.
The fact that the upperbound $b_k$ is an increasing function of $k$ is in accordance with the idea that the permissiveness of the strategies increases when $k$ increases.

We discuss next which strategy $S_x$ should be chosen so as to optimize resource utilization.

It is assumed that a process does not use all of the resources it claimed all of the time and so it uses, on the average, only a fraction of its claim. This means that an arbitrarily given set of competing processes could use a fraction $f$ of its total claim, amounting to $f \times \eta_1$, if all restrictions (such as the number of resources actually available) were entirely disregarded. With regard to optimal resource utilization, the set of competing processes should, for a given $f$, be chosen such that $\eta_1$ (the total claim) is large enough so as to create a situation that indeed requires at least $\text{tot}$ resources, i.e. $\eta_1$ should be chosen such that

$$f \times \eta_1 \geq \text{tot}$$

Conversely, given a set of competing processes and so given a value for $\eta_1$, we can determine which fraction of the total claim should at least be used so that we really need that many resources. Since the choice of a strategy $S_k$ limits the total claim $\eta_1$, it also determines an interval for $f$ for which it makes sense to have as many as $\text{tot}$ resources. We found that for strategy $S_k$

$$\max (\eta_1) = b_k \times \text{tot}$$

and so $f$ should have a value such that

$$f \times \max (\eta_1) = f \times b_k \times \text{tot} \geq \text{tot}.$$ 

Thus, the interval associated with strategy $S_k$ is

$$\frac{1}{b_k} \leq f \leq 1,$$

i.e., for values of $f$ in this interval, strategy $S_k$ admits enough processes to the set of competitors so as to justify the availability of as many as $\text{tot}$ resources.

23.
If strategy $S_0$ is adopted, we find that the interval shrinks into
the single value $f = 1$, because $b_0 = 1$. This says that in the
conservative policy all the claimed resources are supposed to be in use.
This matches nicely with what we found at the end of section 4, viz
that the claimed resources can be allocated right away when the
conservative strategy is applied.

The distinction between the conservative strategy and $S_1$, the
next closest policy, is tremendous. $S_0$ is the suitable policy only
in case all processes use their claimed resources all the time, but
$S_1$ takes advantage of all situations in which the utilization is some-
what less than the total claim and it is the suitable policy for all
cases in which on average just over half of the total claim is
actually in use.

Another significant conclusion is that policies $S_1$ and $S_2$ are
likely to be adequate irrespective of the number of resources.
We see a rapid fall of $f_k = \frac{1}{b_k}$ in going from $k = 0$ to $k = 1$, but subsequent strategies do not widen the intervals in which $f$ is constrained significantly. This is also demonstrated in the table at the end of this section. It seems reasonable to expect that the claims of the processes are quite realistic, so we can assume that the number of resources actually in use is a considerable fraction of their claim. A usage of about half or one third of the claim is probably a low figure and a usage of only one fifth of the total claim for all the processes together is expected to be rather extraordinary. Such a peculiar situation would occur if processes normally use a certain number of resources, but they need once in a while, and only for a short period of time, five times as many resources as usual without being able to release the resources they normally hold. If we assume that it is not likely that all competing processes show such behaviour, then we come to the conclusion that, for instance, $.3 < f < 1$ is a suitable range for $f$. This range even allows the situation that all processes claim up to three times as many resources as they actually need. So we conclude that in normal circumstances a strategy $S_k$ where $k = 1, 2$ or perhaps $3$ is perfectly adequate independent of the value of $\text{tot}$. A more detailed test as in the permissive strategy seems to be of little value. Generally speaking we conclude that the choice of strategy ought to depend on the desired interval for $f$ rather than on the value of $\text{tot}$. But this then reduces the tests to a fixed number of comparisons independent of either the number of resources or the number of competing processes.

The diagram below shows how the safe and unsafe states are related to the $f$-intervals in example E6 where $\text{tot} = 4$. 

25.
<table>
<thead>
<tr>
<th></th>
<th>in use 4 safe</th>
<th>in use 4 not safe</th>
<th>in use 3 safe</th>
<th>in use 3 not safe</th>
<th>in use 2 safe</th>
<th>in use 2 not safe</th>
<th>in use 1 safe</th>
<th>in use 1 not safe</th>
<th>Total safe</th>
<th>Total unsafe</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{1}{2} \leq f \leq 1$</td>
<td>32</td>
<td>11</td>
<td>24</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>64</td>
<td>11</td>
</tr>
<tr>
<td>$\frac{1}{2} \leq f &lt; \frac{1}{2}$</td>
<td>6</td>
<td>22</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>13</td>
<td>26</td>
</tr>
<tr>
<td>$\frac{1}{2} &lt; f &lt; \frac{1}{2}$</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>38</strong></td>
<td><strong>40</strong></td>
<td><strong>26</strong></td>
<td><strong>8</strong></td>
<td><strong>11</strong></td>
<td><strong>1</strong></td>
<td><strong>4</strong></td>
<td><strong>0</strong></td>
<td><strong>62</strong></td>
<td><strong>34</strong></td>
</tr>
</tbody>
</table>

The table shows that there is not much to be gained in going much further than $f \geq \frac{1}{2}$, particularly if almost all resources are in use. The first range corresponds roughly to strategy $S_1$, the first and second together to strategy $S_2$, and all three together to strategy $S_3$. 
7. The programming of strategy $S_1$.

Strategy $S_1$ requires that $\eta_2 \leq \text{tot} - 1$ or, after expansion into a sum of the number of processes per category:

$$n_2 + 2n_3 + \ldots + (\text{tot} - 1)n_{\text{tot}} \leq \text{tot} - 1$$

(where $n_i$ is the number of processes whose claim = $i$ and tot is the total number of processes).

We saw in section 6 that it is not necessary to check $N_2$, $N_3$, ..., $N_{\text{tot}}$, nor $P_2$, $P_3$, ..., $P_{\text{tot}}$ if $\eta_2 \leq \text{tot} - 1$. So, only $N_1$ and $P_1$ remain to be checked.

$$N_1 = \sum_{i=1}^{\text{tot}} n_i, \ \text{so} \ N_1 = \text{number of admitted processes}$$

$$P_1 = \sum_{i=1}^{\text{tot}} p_i, \ \text{so} \ P_1 = \text{number of allocated resources}$$

Strategy $S_1$ can be programmed using three global variables representing respectively $\eta_2$, $N_1$ and $P_1$, and two individual variables for each participating process, representing their claim and rank. None of these variables has to be accessible to any of the competing processes, not even their claim or rank. The monitoring system can properly maintain all of the information if a process communicates its claim to it when starting in the competition.

Updating and checks have to be carried out in four places:

- entrance, when a process requests its first resource;
- request, when a process requests further resources;
- release, when a process returns a resource;
- exit, when a process returns its last resource.

We discuss briefly what has to be done at each of these four places.

Entrance. A process calls enter($x$) when it wishes to be admitted to the set of competing processes. The value parameter $x$ transmits the claim of the calling process. This value must be in the range $0 < x \leq \text{tot}$ so as to ensure that the state is realizable. We will assume that the index $i$ of the calling process is implicitly passed with the call.
The framework of \text{enter}(x)$ is:

- check $0 < x \leq \text{tot}$
- check that \text{rank}_i$ and \text{claim}_i$ are not yet initialized
- initialize \text{rank}_i$ and \text{claim}_i$ at $x$
- increment $\text{N}_2$ by $(x-1)$ and test $\text{N}_1 \leq \text{tot}$
- increment $\text{N}_1$ by 1 and test $\text{N}_1 \leq \text{tot}$
- \textbf{if} either one of the last tests is unsuccessful \textbf{then} undo what was changed and link caller onto ending entrylist \textbf{else} call request.

Request. This function does not need a parameter if the processes request their resources one by one. An extension of the necessary work in case a process is allowed to ask for more than one resource at a time is straightforward and will not be discussed here.

The procedure request is essentially:

- decrement $\text{P}_1$ by one and test $\text{P}_1 \leq \text{tot}$
  (this is the same as testing that the number of remaining resources does not become negative)
- \textbf{if} the last test is unsuccessful \textbf{then}
  undo what was changed and link caller onto requestlist

Release: does not require a parameter if resources are returned one by one. Its task is:

- increment $\text{rank}_i$ by one and check $\text{rank}_i \leq \text{claim}_i$
- decrement $\text{P}_1$
- select and remove a process from the request list (if any)
- \textbf{if} one found \textbf{then} restart its request
- \textbf{if} $\text{rank}_i = \text{claim}_i$ \textbf{then} call exit.

Exit: does not have to be available as a monitor function. The only place where it is called from is within release. Its function is:

- decrement $\text{N}_2$ by one
- decrement $\text{N}_2$ by $(\text{claim}_i - 1)$
- reset $\text{rank}_i$ and $\text{claim}_i$ to uninitialized
- select and remove a process from entrylist (if any)
- \textbf{if} one found \textbf{then} restart its call on enter.

(\text{It is assumed that a process in the entrylist has its claim value attached to it}).
The most frequently executed procedures are request and release. Unlike deadlock avoidance algorithms that are based on claims and number of allocated processes, the tests are independent of the number of resources or number of processes. As a matter of fact the tests are hardly more complicated or time consuming than testing the realizability of an allocation state.

If the resource utilization is expected to be less than about half of what the processes together claim, it may be better to apply strategy $S_2$ or $S_3$. The monitor functions can easily be modified to handle either of those strategies. Adopting $S_2$ implies testing $\eta_3 \leq \text{tot} - 2$ instead of $\eta_2 \leq \text{tot} - 1$. It is then also necessary to check $N_2$ and $P_2$ in addition to $N_1$ and $P_1$. For this purpose, however, only two more variables are needed to record $N_2$ and $P_2$. The change in the programs amounts to updating and checking the pair $N_1$, $N_2$ rather than $N_1$ and the pair $P_1$, $P_2$ instead of only $P_1$. The price to be paid in terms of data maintenance and system overhead is still very low in any of the simple strategies close to the conservative policy $S_0$.

It is conceivable to design an adjusting strategy depending on the number of participating processes. The idea is this: suppose in a given situation there is an element $\eta_k \leq \text{tot} + 1 - k$, so elements $P_k$, $\ldots$, $P_{\text{tot}}$ and $N_k$, $\ldots$, $N_{\text{tot}}$ don't have to be checked. Suppose a process requests admission, but granting admission would imply that $\eta_k \leq \text{tot} + 1 - k$ does not hold any longer. Instead of denying this process admission, we could test for which $i > k$, after admitting this process, $\eta_i \leq \text{tot} + 1 - i$ will be true. From that moment onwards we must check the elements $P_k$, $\ldots$, $P_{i-1}$ in the appropriate places. When a process exits, we test whether there is now a $j < i$ for which $\eta_j \leq \text{tot} + 1 - j$ and if so, $j$ will be used next as the stop criterion for testing $P$ or $N$. Such an adjusting strategy has certain nice properties and would be preferable to earlier deadlock avoidance methods. One could, for instance, attach a certain cost for joining the set of competitors based on what the value of $k$ is going to be for which $\eta_k \leq \text{tot} + 1 - k$. The advantage over classic deadlock avoidance algorithms is that testing of $P$ and $N$ is terminated at the earliest moment possible.
Compared to the fixed strategies discussed in this paper, however, the adjusting strategy has a great disadvantage. If adjusting will take place frequently, it becomes almost necessary to maintain the vectors $\mathbf{P}$ and $\mathbf{N}$ entirely, because any part of those may be needed in the near future. This, however, implies that the tests are back to linear again, proportional to the number of resources. If on the other hand, adjustment is not likely to occur very often, then we can argue that the vectors $\mathbf{N}$ and $\mathbf{P}$ don't have to be recorded entirely, because in that case we can afford to reconstruct the elements of $\mathbf{P}$ and $\mathbf{N}$ that are needed from the current values of ranks and claims. However, a fixed strategy probably works as well as an adjusting policy if adjustment is a rare event.

Conclusion.

A safety test against system deadlocks does not have to be more complex than a small number of increment-and-test instructions. There are strategies between the most conservative and the most permissive policies that have almost all the advantages of either of these extreme strategies without sharing in much of their disadvantages. The advantages are a fixed number of tests instead of a number proportional to the number of processes or the number of resources, and acceptance of the majority of safe allocation states that are likely to occur. A strategy close to the conservative policy is adequate in normal circumstances.

The discussions in this paper dealt with the case of many resources of one kind. A next step in these matters is an investigation of the impact that these results have on the situation of more than one kind of resource. It is expected that the tests will have to be proportional to the number of different resource types involved. An adjusting strategy as presented in the last section, however, may perform much better than that.

The motivation for applying a deadlock avoidance strategy is higher resource utilization and improvement of throughput. Though the examples certainly give evidence that these goals will be achieved, it remains to be shown in an analysis (producing hard figures) whether there is a valid
argument. We have concluded in this report that a safety test can hardly be rejected on the grounds of system overhead.

Another aspect that requires further investigation is the probability distribution of safe and unsafe states given a distribution of the number of processes that start with the same claim. This information is probably sufficient to determine the value of the utilization fraction \( f \). Once this number is computable, we can also find the optimal strategy associated with it. Such a study is expected to confirm that in many resource allocation systems a policy close to the conservative strategy suffices.

References.


