ANALYSIS OF DEADLOCK AVOIDANCE SCHEMES AND RESOURCE UTILIZATION FOR NON-PREEMPTIBLE RESOURCES

Hsiau-chung Chang

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa.

June 20, 1975

Submitted to Carnegie-Mellon University
in partial fulfillment of the requirement
for the degree of Doctor of Philosophy

ANALYSIS OF DEADLOCK AVOIDANCE SCHEMES AND RESOURCE UTILIZATION FOR NON-PREEMPTIBLE RESOURCES

Hsiau-chung Chang
Department of Computer Science
Carnegie-Mellon University

Thesis Abstract

Two aspects of the performance of deadlock avoidance schemes are studied. The first is the cost of deadlock avoidance algorithms. This represents the system overhead. The second is the resource utilization under different schemes. For the first, the basic cost is the computation of the boolean function safe(I), where I is an integer vector representing the system state. safe(I) is true if the system is safe, false otherwise. The resource allocator will make the allocation only when the resulting system has safe(I)=true. Based on the concept of computation trees, several lower bounds for the cost involved in computing safe(I) are established under different conditions. An upper bound is also developed.

If the special characteristics of the system enable one to have a relation $\text{safe}(I) \equiv C_p(I) \wedge C(I)$, such that $C_p(I)$ is always true, the cost of computing safe is reduced to that of computing C. The control system may also impose certain condition C_p, to have $C_p \rightarrow (\text{safe} \leftrightarrow C)$. C_p is the admission condition. No initial allocation is made if C_p is violated. The effective cost of computing safe is again reduceable. Several schemes using such reductions are discussed. A number of properties of the system are also discussed.

When external priorities exist, the safety check takes a slightly different form. It is shown that the existence of the external priorities, as far as the computation cost is concerned, is equivalent to the existence of one more type of resources.

The resource utilization, defined as the average number of busy resources, is studied under different admission conditions C_p. For small systems exact solutions are obtained. For large systems an approximate method is developed. The result is compared with the result of simulations. The study shows that i) the basic resource requirement of each process must be recognized. ii) an over-populated system is much worse than an under-populated system iii) the number of suspended processes should be in the range 0~1. Also studied are the transient behaviors and the activation disciplines. From these studies it can be deduced that it is possible to increase the resource utilization and decrease the system overhead at the same time by imposing appropriate admission conditions.

TABLE OF CONTENTS

CHAPTER 1

Introduction

1.1. Non-preemptible Resource Allocation Problem

This is a condensed version of the thesis. The problem will be discussed and results will be given, but no proof is included. There is no survey of the literature, either. (They may be included later) As a whole, this should be viewed as a summary of the results obtained in the thesis.

The basic problem to be studied is the non-preemptible resource allocation problem. There are two concerns for such problem: The cost of the deadlock avoidance schemes and the resource utilization. For the first, since there is a real danger of getting into the deadlock, the control system must make sure that the system is always in a safe state. (to be defined later) Before making each allocation, the control system must check whether the system is still in a safe state after the allocation, and the allocation will be made only when the resulting state remain safe. This is called the safety check. As the system is progressing, a large number of safety checks have to be carried out, and this represents a significant portion of the system overhead. We are interested in the cost of doing it. For the second, a high resource utilization is a goal every system designer likes to achieve. The deadlock problem is a consequence of our desire to increase the utilization by encouraging the resource sharing among processes. We want to know whether the goal of increasing the resource utilization is indeed achieved.

The thesis is divided in two parts. The first part is a study of safety conditions and the cost of doing the safety check. The cost is defined to be the number of comparisons, and the concept of the computation tree is used. Several lower bounds and an upper bound for the safety check are established under various conditions. Some properties of the safety condition are further explored, to enrich the possibility of making use of known conditions about the system state. Also studied is a scheme to include the priority discipline. The second part is the study on the resource utilization. Several smaller examples are first studied, to get some initial understanding about the system behavior. An approximate method is developed to solve larger systems. Among the most important results are i) The request of a process must be recognized. ii) The over-populated system is much worse than the under-populated system. iii) The number of suspended processes should be in the range of 0~1 to get good utilization.

1.2. More Formal Description

There are two basic entities, the processes and the resources. The control system will allocate the resources to processes according to their demands. The process will return the resources to the control system when it finishes, but the control system may not take the resources back by force. In other words, the resources are non-preemptible.

There are $\underline{tot}$ resources of n types in the system. $\underline{tot}$ is a vector of n components. Each component is the total number of resources of a particular type. A process is described by $P=(\underline{c}, \underline{a}, ,\underline{d})$. $\underline{c}$ is the claim, which is the maximum number of resources it ever needs. $\underline{a}$ is the allocation, which is the number of resources allocated to the process. $\underline{d}$ is the demand, which is the number of resources the process is asking for, but has not been allocated yet. It will be shown that $\underline{d}$ is irrelevant as far as the safety condition is concerned. For the first part of the thesis, we may use ($\underline{c}$, $\underline{a}$), or other equivalent form, to represent a process. Two other quantities are defined for a process. The rank, $\underline{r}=\underline{c}-\underline{a}$, and the request, $\underline{q}=\underline{a}+\underline{d}$.

The competitor set is the set of processes P_i with $\underline{a}_i \neq 0$. It is assumed there are m processes in the competitor set. Apparently the system safety depends only on the competitor set. m, the number of processes in the competitor set, and n, the number of types, will appear as parameters in the later algorithmic study. The remainder, $\underline{rem}$, is defined to be

$$\underline{rem}=\underline{tot}-\sum_{i=1}^{m} \underline{a}_i.$$

and is the number of available resources.

The following notations are often used.

ρ: a sequence of distinct integers

$\rho(i)$: the i'th integer of ρ

$|\rho|$: length of ρ

$i \in \rho$: the integer i is in ρ

1.3. Safety Condition

The system safety is based on the notion that, if in the worst condition, all processes can run to completion one by one according to certain order, then the system is safe. Otherwise the system is not safe. This is formally defined as follows.

A partial completion sequence, PCS, is a sequence ρ such that

$$\underline{r}_{\rho(i)} \leq \underline{rem}+\sum_{j=1}^{i-1} \underline{a}_{\rho(j)} \qquad \text{for } 1 \leq i \leq |\rho|$$

ρ is just a sequence according to which several processes can run to completion in the worst case. A process, P_i, is safe if in the worst condition it can run to completion, i.e., there exists a PCS, ρ, such that $i \in \rho$.

A completion sequence, CS, is a sequence ρ such that ρ is a PCS and $|\rho|=m$.

A completion sequence is the order according to which all processes can run to completion in the worst case. A system is said to be safe if there exists a completion sequence. Otherwise it is unsafe. The safety check is that, given a system description, to determine whether the system is safe or not.

There are two existing algorithms for doing the safety check: Habermann's algorithm takes $nm(m+1)/2$ comparisons in the worst case. Holt and Russel's algorithm takes $2nm+n \log_2 m!$ comparisons in the worst case. In chapter 2 the number of comparisons will be defined to be the cost of the safety check algorithms, and these numbers may be referenced. Through out this thesis, the logarithms will be assumed to have base 2.

CHAPTER 2

The Cost of Safety check

2.1. The Computation Tree

The basic operation involved in the safety check is the comparison. Hence the number of comparisons will be defined as the cost of computation. The costs of two existing algorithms were shown in Chapter 1. In this chapter we will establish several lower bounds and an upper bound under different conditions. These in turn show that the current algorithms are indeed very good.

For simplicity, we will use an integer vector $\underline{x}$ to collectively represent $\underline{rem}$, $\underline{a}_i$'s and $\underline{r}_i$'s as follows.

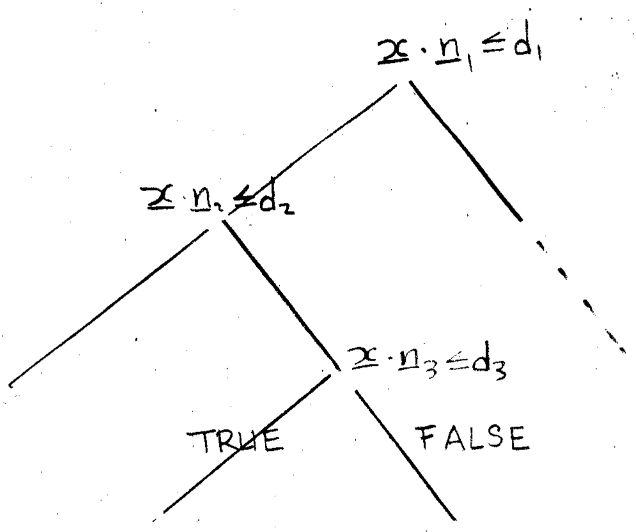
$\underline{x}=(\underline{rem},\underline{r}_1,\underline{r}_2 \cdots \underline{r}_m,\underline{a}_1,\cdots \underline{a}_m)$

$\underline{x}$ can be thought of as the input to the safety check algorithm. Note that the system safety depends only on $\underline{x}$.

The development is based on the concept of the computation tree, as studied by Reingold. For the purpose here, a computation tree is defined to be a binary tree such that

i) Associated with each node is a comparison, or condition, of the form $\underline{x}\cdot\underline{n}\leq d$, where $\underline{x}$ is the input and $\underline{n}$ and d are constants.

ii) Associated with each terminal leaf is an value, which is the result of the computation. The value can be TRUE, FALSE, a number, a sequence, or anything with agreed upon interpretation. A typical computation tree is shown in fig. 2.1.1. The


$\underline{x} \cdot \underline{n}_1 \leq d_1$
$\underline{x} \cdot \underline{n}_2 \leq d_2$
$\underline{x} \cdot \underline{n}_3 \leq d_3$
TRUE
FALSE

node ·	linear comparison
leaf	value

Computation Tree

fig. 2.1.1

height of the tree is the longest distance between the root and the leaf. The width of the tree is the number of leaves. The cost of the algorithm is defined to be the height of the tree. This actually is the number of comparisons in the worst case.

Each input can be thought as an token travelling down the tree from node to node until reaching a leave, and the value associated with that leave is the value of the computation with that input.

2.2. Results Based on Linearly Independent Conditions

Two results are based on the theorem that, in order to verify q linearly independent conditions, at least q comparisons are necessary. From this theorem, we have

i) The verification of a CS takes at least nm comparisons. That is, suppose there is a particular ρ. The algorithm which will accept an input and determine whether ρ is a CS for that input, takes at least nm comparisons.

ii) The safety check takes at least nm comparisons. That is, to verify whether there exists a CS or not, at least nm comparisons are necessary.

2.3. Result From The Property of Convex Set

The above results are easy to prove, but are not sharp. The following is an improved bound for single type resources, i.e. for $n=1$

i) To look for a CS, at least log m! comparisons are necessary: A practical approach of checking whether the system is safe or not is to explicitely look for a CS. For such algorithms, at least log m! comparisons are necessary. This proof is based on the theorem that height≥log width.

ii) For any safety check, at least log m! comparisons are necessary. This says that no matter what the approach is, at least log m! comparisons are necessary. The proof is based on the property of the convex set.

2.4. Upper Bound

It is possible to build a computation tree for safety check with height equal $m(n-1)+m(m+1)/2$. This is an improvement over both algorithms discussed in Chapter 1 for large values of n. It essentially says that whenever the number of types is increased by 1, the cost is increased only by m. The algorithm is a modification of Habermann's algorithm. It keeps track of all compared ranks, so that a rank is seldomly compared twice.

2.5. Lower Bound for Type-Restricted Algorithms

An algorithm for the safety check is said to be type-restricted if it does not compare quantities of different types. As a matter of fact, no known safety check algorithms involve comparisons among different types, and it is difficult to conceive any algorithm which can profit by doing so. For type restricted algorithms, it is shown that at least $(n-1)m + \log m!$ comparisons are necessary.

CHAPTER 3

Equivalent Conditions and Other Properties

3.1. The Prior Conditions

Sometimes we can have the condition

$C_p(\underline{I}) \wedge C(\underline{I}) \equiv \mathrm{safe}(\underline{I})$

where $\underline{I}$ is the system state and safe($\underline{I}$) is a boolean function with parameter $\underline{I}$. safe($\underline{I}$) is true if $\underline{I}$ is safe, false otherwise. If for some reason $C_p(\underline{I})$ is true, then we can compute safe($\underline{I}$) by computing $C(\underline{I})$ only. The effective cost of safety check can thus be reduced. $C_p(\underline{I})$ may come from our knowledge about the system state, or from the partial result saved from the earlier computation. Sometime we can even impose certain artificial condition on the system to have

$C_p(\underline{I}) \rightarrow (\mathrm{safe}(\underline{I}) \leftrightarrow C(\underline{I}))$ (1)

Here $C_p(\underline{I})$ is the admission condition. By maintaining the system so that $C_p(\underline{I})$ is always true, the cost of safety check again can be reduced.

The success of splitting safe($\underline{I}$) into two components depends on our knowledge about the system. A number of properties are listed below. Many of them represent useful knowledge about the system, and can be used to realize the division of safe($\underline{I}$).

3.2. Connected Processes

Two processes P_1 and P_2, are said to be connected if there exists a $\underline{k}$ such that $\underline{r}_1 \leq \underline{k} \leq \underline{c}_1 \wedge \underline{r}_2 \leq \underline{k} \leq \underline{c}_2$. The phrase "connected" refers to the fact that the graph representation of two processes shows overlapped rectangles.(see thesis) Several processes are mutually connected if we can "travel" from one process to another one. We can recursively define that P_1 and P_2 are connected if there exists a P_3 such that P_1 and P_3 are connected, and that P_2 and P_3 are connected.

Several properties are:

i) If P_i is safe, then all processes connected to P_i are safe.

ii) If all processes have the same claim,(so that they are connected), then the safety of any process implies the system safety. This means that if all processes have the same claim, then the safety check algorithm can terminate as soon as a process is found safe. Another way to say it is that if a process is safe, then all processes of the same claim are safe. When there are only two or three different claims for all processes, this is applicable.

iii) If there exists a process P_j such that $\underline{r}_j \le \underline{r}_i$ for all i, then

P_j is safe$\leftrightarrow$the system is safe

iv) For single type resources, r_j in iii) always exists. Hence, only one comparison is necessary.

3.3. Parallel Conditions

Several equivalent conditions for the safety are shown below.

Th6: The safety condition is equivalent to the following condition: There exists a sequence of $\underline{k}_i$'s, $0 \le i \le q$, such that

$$\underline{k}_0 = \underline{0} \le \underline{k}_1 \le \ldots \le \underline{k}_q = \underline{tot} \text{ and}$$

$$\underline{k}_{i+1} \le \underline{tot} - \sum_{\underline{r}_j \nleq \underline{k}_i} \underline{a}_j$$

Let $\underline{u}_j$ be the unit vector in j'th direction, i.e., $\underline{u}_j=(0,0,0,...1,0,0..)$. The above theorem can take another form.

Th7: The safety condition is equivalent to the following condition: There exists $\underline{k}_i$'s, $0\le i\le q$, such that

$\underline{k}_0=\underline{0}$, $\underline{k}_{i+1}=\underline{k}_i+\underline{u}_j$ for certain j, $\underline{k}_q=\underline{tot}$, and

$\underline{k}_{i+1}\le\underline{tot}-\sum_{\underline{r}_l\nleq\underline{k}_i}\underline{a}_l$

This can be further modified as follows.

Th10: The safety condition is equivalent to the following condition:

For any $\underline{k}$ such that $\underline{0}\le\underline{k}\le\underline{tot}$, $\underline{k}\ne\underline{tot}$,

$\underline{tot}-\sum_{\underline{r}_l\nleq\underline{k}}\underline{a}_l\nleq\underline{k}$

These are actually $tot_1\cdot tot_2\cdot..tot_n$ conditions with no explicit relations among each other. This condition can be more generalized as follows.

Th13: S is safe$\leftrightarrow$

For any $\underline{k}$ and $\underline{r}_i'$, $1\le i\le m$, such that

$\underline{o}\le\underline{k}\le\underline{tot}$ and

$(\underline{r}_i'>\underline{k}\wedge\underline{r}_i\nleq\underline{r}_i'\le\underline{c}_i)\vee(\underline{r}_i'=\underline{c}_i)$,

$\underline{tot}-\sum_{i=1}^{m}(\ge c_i-\underline{r}_i')\nleq\underline{k}$

The equivalent condition in Th13 can also be stated as: for any $\underline{k}$ such that $\underline{o}\le\underline{k}\le\underline{tot}$ and $\underline{r}_i'$, $1\le i\le m$,

$\underline{tot}-\sum_{i=1}^{m}(\underline{r}_i'\nleq\underline{k})(\underline{r}_i'\le\underline{c}_i)(\underline{r}_i\le\underline{r}_i')(\underline{c}_i-\underline{r}_i')\nleq\underline{k}$

For single type resources, this result specializes to the result of Habermann's. We can drop the vector notation, and replace $\nleq$ by >. The safety condition is then

$tot-\sum_{i=1}^{m}(r_i'>k)(r_i'\le c_i)(r_i\le r_i')(c_i-r_i')>k \quad (1)$

for all $0\le k<tot$ and any r_i'

These can be used to deduce Habermann's conditions.

Let us define $|\underline{v}|$, for any vector $\underline{v}$, to be the sum of components of $\underline{v}$, i.e.,

$$|\underline{v}|=\sum_{i=1}^{n} v_i$$

The relation $|\underline{v}_1|>|\underline{v}_2|\rightarrow\underline{v}_1\not\leq\underline{v}_2$ is obvious. This can be used to deduce that

Th16: If, for all $\underline{k}$ such that $\underline{0}\leq\underline{k}\leq\underline{tot}$ and $\underline{k}\neq\underline{tot}$, $\sum_{\underline{r}_j\not\geq\underline{k}} |\underline{a}_j|<|\underline{tot}|-|\underline{k}|$, then the system is safe.

3.4. Partial Reservation

Consider the following condition:

$$\sum_{i=1}^{m}(\underline{c}_i-\underline{k})\leq\underline{tot}-\underline{k}$$

for certain $\underline{k}$. This form is inspired by the study in the previous condition. The intuitive explanation is, if on the average a process uses $\underline{c}_i-\underline{k}$ of the claimed resources, then we just reserve $\underline{c}_i-\underline{k}$ resources for each processes. The total reserved resources are then $\sum_{i=1}^{m}(\underline{c}_i-\underline{k})$. However, if for each process P_i $\underline{c}_i-\underline{k}$ resources are allocated to it, then all processes would be at rank $\underline{k}$. In the worst condition, when all processes suddenly request all claimed resources, $\underline{k}$ resources must be available, to avoid the deadlock. Hence, $\underline{k}$ resources should be reserved for the sake of system safety. The number of total reserved resources is then $\underline{k}+\sum_{i=1}^{m}(\underline{c}_i-\underline{k})$. This number must be less or equal then $\underline{tot}$, as indicated. When there are also processes with rank$\not\geq\underline{k}$, the above condition should be modified to

$$\sum_{i=1}^{m}(\underline{c}_i\geq\underline{k})(\underline{c}_i-\underline{k})\leq\underline{tot}-\underline{k} \quad (c1)$$

Several properties are these:

Th18: Under condition c1, if $\underline{c}_i\geq\underline{k}$ and $\underline{r}_i\geq\underline{k}$ for all $1\leq i\leq m$, then the system is safe.

Th19: Consider a system under condition c1. If $\underline{c}_i\geq\underline{k}$ for all i, and there is a q such that

$\underline{r}_i \geq \underline{k}$ if $i \neq q$, then the system is safe.

Another property offers early termination conditions.

Th20: Under condition c1, if

$\underline{r}_i \ngeq \underline{k} \rightarrow P_i$ is safe for all $1 \leq i \leq m$

then the system is safe.

This can be specialized in two ways:

Th21: Consider single type resources. Under c1, if all processes whose ranks are less than k are safe, then the system is safe.

Th22: Under c1, if $\underline{r}_i \leq \underline{rem}$ for all $\underline{r}_i \ngeq \underline{k}$, then the system is safe.

CHAPTER 4

Priorities

4.1. Permanently Blocked Processes and External Priority

Although the system deadlock can be avoided by using certain deadlock avoidance schemes, there is no guarantee that a process may be activated in a reasonable length of time. This was first pointed out by Holt. It can be shown that this is actually one facet of the priority problem. The priority may be in conflict with the safety requirement, because the priority may dictate that the processes be activated according to a certain order, yet the safety requirement demands the other way.

To solve this problem, we have to recognize the existence of the priority at the first place, and to define an overall safety criterion. Let G(i) be the priority of the process P_i. $G(i)<G(j)$ means P_i has higher priority then P_j. We can have the following modified definition for safety.

def: The system is G-safe is there exists a CS, ρ, such that $G(\rho(i))\leq G(\rho(i+1))$ for all $1\leq i\leq m-1$

It can be proved that, as far as the cost of the safety check is concerned, the cost of G-safety is the same as that of safety check with one more type of resources.

The application of the G-safety is this. Suppose P_k has been waiting too long and has to be activated. Let ρ be a CS. We can assign a G such that $G(\rho(i))=0$ if $i\leq\rho^{-1}(k)$, $G(\rho(i))=1$ if $i>\rho^{-1}(k)$. In other words, there are only two different priorities. Processes of higher priority are free to run, being subjected only to the safety requirement. Processes of low priority are able to run only when they do not demand more resources.

The advantage of this scheme is that G can be dynamically reassigned. Suppose we have already assigned a G according to certain ρ. This means that in the subsequent allocations, only G-safe allocations will be made. However, to check the G-safety usually involves looking for a G-CS. If the G-CS, ρ', is found such that $\rho'^{-1}(k)<\rho^{-1}(k)$, then we can reassign another G' such that $G'(i)=0$ if $i \le \rho'^{-1}(k)$, and $G'(i)=1$ otherwise. P_k is thus promoted forward. This reassignment is possible, because we always have $\rho'=\rho_1\rho_2$ such that $G(\rho_1)=0$ and $G(\rho_2)=1$. The new assignment of G' does not cause any conflict. This is an improvement over earlier suggested schemes.

At any moment of assigning a G, the following properties can be used to forwarding $\rho^{-1}(k)$

Th8: Let $\rho=\rho_1\rho_2$ be a G-CS, such that $G(\rho_1)=0$, $G(\rho_2)=1$. If $1 \le i<j \le |\rho_1|$ and $\underline{r}_{\rho_1}(j) \le \underline{r}_{\rho_1}(i)$, then there exist a G-CS ρ', $\rho'=\rho_1'\rho_2$, such that $\rho_1'(i)=\rho_1(j)$

CHAPTER 5

Resource Utilization for Small Systems

5.1. Process Description

This and the following chapters are on the resource utilization study. Only single type resources will be considered, and the systems are assumed to be Markovian. Each process has a claim, c, and a request, q. q is the number of resources the process needs at certain moment for running. All quantities, a, r, d, are as defined before. The state of a process is identified by q. When a process is running freely, its behavior is described by λ_{ij}, the transition rate from q=i to q=j. fig. 5.1.1

$$\begin{tikzcd}(q=1) \arrow[r, dash, dashed] & (i-1) \arrow[r, shift left, "\lambda_{i-1\,i}"] & (i) \arrow[l, shift left, "\lambda_{i\,i-1}"] \arrow[r, shift left, "\lambda_{i\,i+1}"] & (i+1) \arrow[l, shift left, "\lambda_{i+1\,i}"] \arrow[r, dash, dashed] & (q=c)\end{tikzcd}$$

fig. 5.1.1

shows the process.

The process, as far as its internal state is concerned, can be imagined as a token wandering from state to state. Due to the assumed Markovian property, we can use transition rate instead of transition probability.

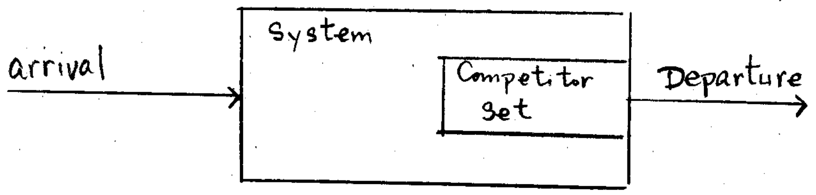
At any state, a process has a constant finishing rate β, i.e., βdt is the probability the process may finish and leave in the time interval (t,t+dt).

The arrival rate of the process is α. For most of the time we will consider only systems under heavy demand. In such systems $\alpha \to \infty$, and we can use a closed network and get rid of α.

f_i, $1 \le i \le c$, are defined to be the probability $P\{q=i\}$ when the process is running alone. This is the probability that a process may request i resources. The average request, $\bar{q}$, is defined to be $\sum_{i=1}^{c} if_i$.

5.2. System Description

The system is shown in fig. 5.2.1



M : max. # of processes in the system
m : max. # of processes in the comp. set

fig. 5.2.1

. The number of resources in the system is tot. The processes with $a \ne 0$ form a competitor set. The process arrives in the system and waits to be admited into the competitor set. Once the process is in the competitor set, the resources allocated to the process can not be taken back by the system by force. A process is active if d=0, suspended if $d \ne 0$. When a process is suspended, all its activities stop, i.e., all transition rates, such as λ_{ij}'s and β, can be thought of as temporarily being zero.

Sometimes the number of processes in the system, (this is not the competitor set), is limited to M. This can be compared to the limited source waiting line model.

When the system is under heavy demand, as may be assumed later, there will be a constant number, M, of processes in the system.

The resource utilization, U, is defined to be the average number of busy resources. Let

$$t_b = \sum_{P_i \text{ active}} a_i$$

$\bar{t}_b$ is the time average of t_b, and is the utilization U.

A is defined to be the number of active process in the system. $\bar{A}$ is the time average of A. If <u>all</u> <u>processes</u> <u>are</u> <u>identical</u>, $\bar{q}\bar{A}=\bar{t}_b$, and we can use either $\bar{A}$ or $\bar{t}_b$ as utilization.

We want to study the variation of $\bar{t}_b$, or $\bar{A}$, under one of two types of conditions.

i) type-1 conditions: We say the system is under strategy S_k if

$$\sum_{P_i \text{ in the comp. set}} (c_i - k) \le tot - k$$

ii) type-2 condition: The number of process in the competitor set (not system) is limited to m.

For identical processes, as often will be the case to be studied, there is a type-2 condition for any type-1 condition, and the problem would be to study $\bar{t}_b(m)$, the variation of $\bar{t}_b$ as m varies.

In summary, M is the maximum number of processes in the system. The process arrival rate is α, departure rate is β. λ_{ij}'s are the transition rate of a process. The utilization of the system is controlled by setting a type-2 condition, (or a type-1 condition) on the competitor set.

α and β are assumed to be the same for all processes. α appears very infrequently in the thesis, and will not appear in this condensed version. Only systems under heavy demand will be discussed here, and there are M processes in the system in this case.

5.3. System with tot=2 under S0 and heavy demand

The system with c=1 is extensively studied in the queueing theory. We look at the next simplest system with tot=2, c=1 or c=2 for all processes. Assume the system is under S_0($\Sigma c_i \leq$tot, no deadlock problem)

The system is this. There are two resources. The system is under heavy demand, so that there is a constant number, M, of processes in the system, and the system can be described by a closed network as shown in fig. 5.3.1

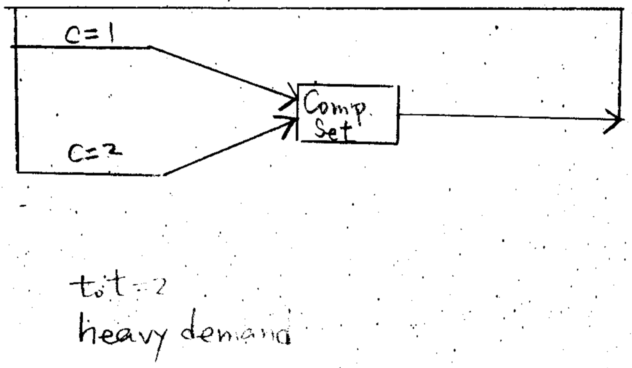


fig. 5.3.1

. For the process with c=2, we have λ_{12} and λ_{21} as transition rates. Under S_0, there is no deadlock problem. The problem is instead how to arrange the mixture of processes in the competitor set to get high utilization.

The results show that the utilization is the worst by giving a higher priority to processes with c=1. The utilization is the best by giving a higher priority to processes with c=2. Let $\lambda_{12}=\lambda_{21}=0.5$. For the first, U=1.42, independent of M. For the second, $U=\frac{10M-10}{6M-5}$, $\rightarrow$1.66 as M is getting larger.

It is also shown that, if a higher priority is given to processes with c=2, the response times of processes with c=1 and processes with c=2 are nearly the same.

5.4. Interference under S1 with tot=2

There is no concurrent execution of two processes with c=1 and c=2 under S_0, but there will be under S_1. We want to know whether it is desirable to do so.

To simplify the matter, we consider the following artificial system. The system is under heavy demand, $M\to\infty$, $\lambda_{12}=\lambda_{21}=\lambda/2$ for 2-processes (processes with c=2). The initial request of a 2-process is 1. Under S_0, 1-process(process with c=1) can never run, and U=1.5. Under S_1, $U=\frac{\lambda/\beta+8}{\lambda/\beta+4}$. To make U>1.5, λ/β should be <4. See fig. 5.4.1

$$\begin{tikzcd}
\begin{pmatrix} 2 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} \arrow[r, "\lambda/2"] & \begin{pmatrix} 2 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \arrow[r, "\beta"] & (2\ 2\ 0) \arrow[ll, bend right, "\lambda/2"']
\end{tikzcd}$$

fig. 5.4.1

This is essentially to show that, once a process is suspended, all resources allocated to it are suspended. Hence, we can not increase the utilization by admitting as many processes as possible into the competitor set. It also shows that the faster the processes' requests are varying, the fewer processes can be admitted into the competitor set. This gives an intuitive explanation for later results.

A simulation is designed and tested for the system with tot=2 and non-zero α and β, and shows that the results obtained in these two sections are good approximations for systems near satuation.

5.5. System with c=2 under type-2 condition and heavy demand $\beta\to 0$

There are a constant number, m, of processes in the competitor set. All processes are identical with c=2. There is no arrival or departure, i.e., $\beta \to 0$. We want to know $\bar{t}_b(m)$.

Let $\lambda = \lambda_{12}/\lambda_{21}$, b_0=tot-m, a_0=2m-tot,

$$P_i = \frac{b_0^i}{i!\,\lambda^i} \quad \text{for} \quad 0 \le i \le a_0$$

$$P_i = \frac{b_0^{a_0} \binom{b_0}{i-a_0}}{i!\,\lambda^i} \quad \text{for} \quad a_0 \le i \le R$$

$$U = \frac{\sum_{i=0}^{m} (i + 2b_i) P_i}{\sum_{i=0}^{m} P_i}$$

where b_i=m-i if 2m-i≥tot, t-m if 2m-i>tot. The result is depicted in fig. 5.5.1

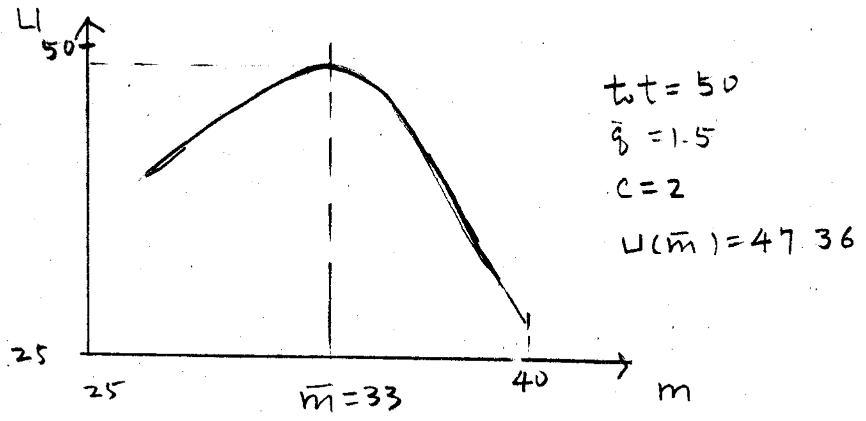


fig. 5.5.1

There are several properties.

i) $\bar{m}$, the optimal value, ~tot/$\bar{q}$

ii) $U(\bar{m}+m) \ll U(\bar{m}-k)$

iii) $\overline{m}-\overline{A}(\overline{m})$ is in the range 0~1, i.e., the number of suspended processes is in the range 0~1.

CHAPTER 6

Utilization for Larger Systems

6.1. Two Approximations

We want to compute $\bar{A}(m)$ as in 5.5, because $\bar{A}$ is an indication of the right mixture of processes in the competitor set. When $c>2$ for all processes, the problem is difficult to solve, even under the assumption that all processes are identical. Here we make two approximations.

i) The approximation for safe condition: If we observe the system over a long period of time, and record the average number of unallocated resources(rem) when there are suspended processes in the system, we can get an idea of how many resources are reserved just for the system safety. These reserved resources are nver allocated. The effective total number of resources is then tot-$\bar{e}$, where $\bar{e}$ is the number of reserved resources. For an observer, the system behaves as if there are only tot-$\bar{e}$ resources, the effective resources. These resources can be allocated as soon as they are demanded. Based on this idea, we can get $e=(mc-tot)/(c-1)$, and $\bar{e}=e/2$.

ii) Pseudo-independent processes approximation: If all processes are independent, say, if $\Sigma c_i \leq tot$ is satisfied and so there is no interference, the we can treat $t_q=\Sigma q_i$ as a sum of m independent random variables, and solve it accordingly. When there are interferences, we can separate a process in two phases, the active phase and the suspended phase. For each phase we designate a random variable for it, and treat all these random variables as independent. This results in a slightly simpler form to handle. (see thesis)

Another assumption is that λ_{ij} is non-zero only when $j=i+1$ or $j=i-1$. This says that the process demands or releases only one resource at a time.

Based on these simplifications, we have the following result, as in fig. 6.1.1

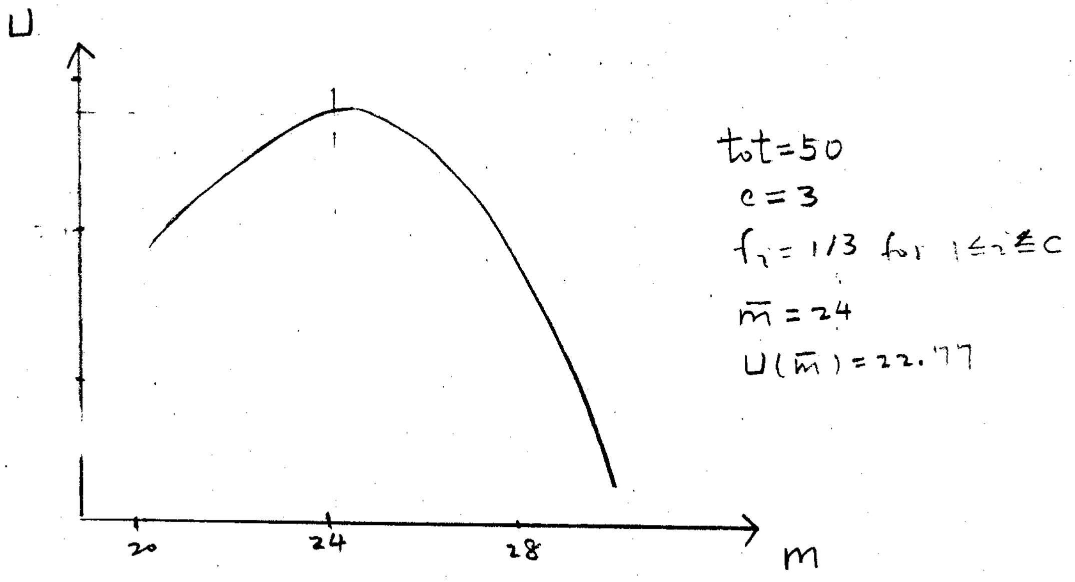


fig. 6.1.1

The same properties as in 5.5. are observed, namely,

i) $\bar{m} \sim (tot-\bar{e})/\bar{q}$

ii) $U(\bar{m}+k) << U(\bar{m}-k)$

iii) the number of suspended processes should be in the range 0~1 when U is optimal.

6.2. Comparison With Simulation

The result in 6.1. is compared with the simulation. In the simulation, we further study the effect of different activation disciplines. Three activation disciplines are studied.

i) First In First Out(FIFO): The first suspended process will be the first one to be activated. This is what 6.1. is studying.

ii) Smallest Allocation First(SAF): When there are several process suspended, and one can be activated, the process with smallest a is activated first.

iii) Largest Allocation First(LAF): Process with largest a is activated first.

The result is shown in fig. 6.2.1

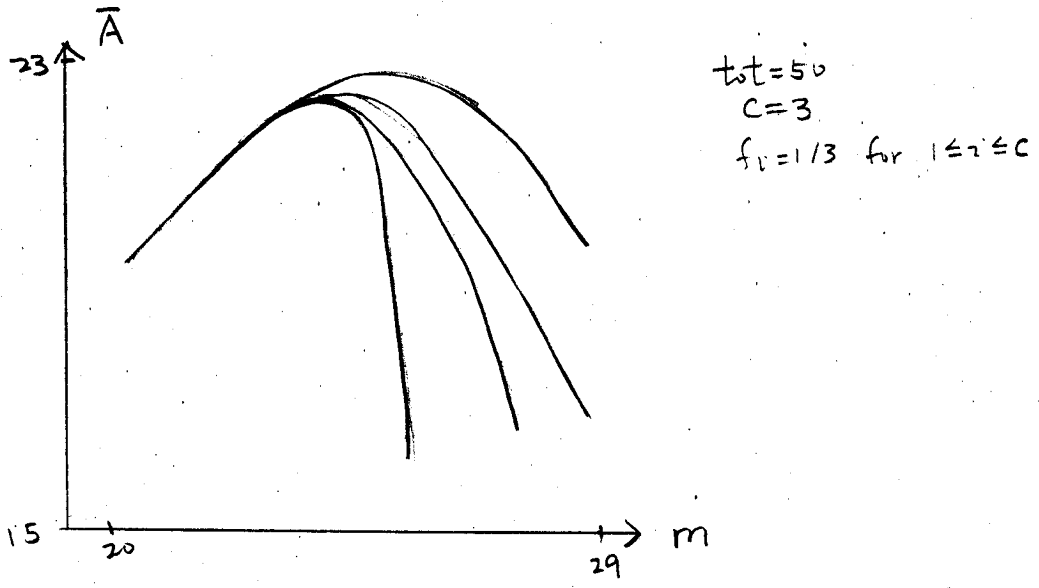


fig. 6.2.1

Two points are shown:

i) The result from the approximate method is quite agreeable with that obtained from the simulation.

ii) LAF is the best and SAF is the worst.