PERFORMANCE ANALYSIS OF THE MULTISAFE PROTECTION ENFORCEMENT PROCESSES

by

Mason C. Deaver, Jr.

Thesis submitted to the Graduate Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE
in
Computer Science and Applications

APPROVED:

L. Ray Hartson
Dr. H. Ray Hartson

Robert B. Frary
Dr. Robert B. Frary

R. W. Ehrich
Dr. Roger W. Ehrich

June, 1983

Blacksburg, Virginia

PERFORMANCE ANALYSIS OF THE MULTISAFE PROTECTION ENFORCEMENT PROCESSES

by

Mason C. Deaver, Jr.

(ABSTRACT)

This paper describes the performance of the MULTISAFE database protection model through response-time equations. A predicate-based protection model is described. Various classes of access decision dependencies are reviewed. The distinct modules of MULTISAFE are discussed, and a relational database approach to the management of data protection is developed for these modules. A performance equation which models user login into MULTISAFE is developed. A set of equations is developed which model the processing of database queries as a series of steps. These equations are then modified to consider the possibility of concurrent processing among the MULTISAFE modules. The two sets of equations are compared and analyzed. The analysis reveals that the concurrency feature of MULTISAFE allows database protection to be implemented with a minimum of system overhead. Further analysis shows that, in some cases, an arbitrary database query takes less time to process with all protection checks in force than a similar query in a protectionless environment.

ACKNOWLEDGEMENTS*

I would like to thank the members of my advisory committee, especially Dr. Hartson, for their valuable assistance. I would also like to thank my fellow members of the Database Research Team, whose combined effort is reflected in this paper. Finally, I would like to thank my wife, Dawn, for her aid, patience and understanding during my tenure at Virginia Tech.

---

*The work reported herein was supported, in part, by the National Science Foundation under Grant Number MCS-7903936.

TABLE OF CONTENTS

Chapter I

INTRODUCTION

There exists a continuing need in computing systems to provide controls on the access of stored data. There have been many system models discussed in the literature to accomplish this task (see, for example, [ASTRM76] and [STONM74]). As noted in [HARTH80], the predicate-based model of protection described in [HARTH76a] was one of the first in which access decisions were sensitive to the state of the system as well as the contents of the data to be accessed. Based on this model of protection, a system architecture called MULTISAFE is being developed. MULTISAFE stands for MULTIprocessor system for supporting Secure Authorization with Full Enforcement for database management. As the name implies, MULTISAFE consists of several logical (and possibly physical) processors which work concurrently to retrieve accessed data and to provide protection of this data. As will be shown later, MULTISAFE derives performance advantages from its concurrency feature. By locating the protection functions of MULTISAFE in a dedicated protection processor (or module), the protection and enforcement mechanisms of the system are isolated from the users and from the operations on the database.

Many people have contributed to the development of the MULTISAFE model, and they have written several papers on different aspects of the model. As mentioned above, [HARTH76a] describes a predicate-based model of protection which provided a springboard for the MULTISAFE model. [HARTH76b] goes beyond the model of [HARTH76a] and describes a more complete, and complex, model of protection. In [TRUER80], intermodule communication in MULTISAFE is discussed. This communication takes the form of procedure calls and returns. The flow of messages and data during protection enforcement within MULTISAFE is modeled with extended Petri nets in [BALLE81]. An implementation model of predicate-based protection in MULTISAFE is discussed in [HARTH80]. The [HARTH80] paper gives specific details on how access decisions are implemented in MULTISAFE. Based on this model, a software simulation project is being developed and is discussed in [TALBT81].

This paper describes the performance of the MULTISAFE model through response-time equations. The time required for the MULTISAFE system to respond to various types of user queries is measured in precise detail in these equations. It is assumed that a typical MULTISAFE user types a database query on a video-display terminal. The details concerning what MULTISAFE does to process and respond to the query are

based on the implementation model in [HARTH80]. Particular attention is given to the many access decisions that MULTISAFE can make, and how long it takes MULTISAFE to make the decisions.

The predicate-based protection model of [HARTH76a] is reviewed in chapter II. In chapter III, various classes of access decision dependencies presented in [BALLE81] and [HARTH80] are discussed. Access conditions imposed on MULTISAFE users by systems administrators may depend on the state of the MULTISAFE system, on the queries submitted by users, or on the data which users are trying to access. The configuration of the MULTISAFE model is discussed in chapter IV. MULTISAFE is divided into several distinct modules. These modules and their functions within MULTISAFE are discussed in the chapter.

In chapter V a relational database approach to protection of data is discussed in detail. This chapter is taken directly from the implementation model of [HARTH80]. It is within this relational framework that users of MULTISAFE submit queries, and that MULTISAFE processes the queries. In chapter VI, various global assumptions are discussed. These assumptions apply to all of the equations developed in this paper. Based on these assumptions, some constants are defined which are used throughout the rest of the paper.

An equation which models the processing of a login request by MULTISAFE is developed in chapter VII. The login equation takes into account the possibility of login failure due to enforcement of an access condition. In chapter VIII, several equations are developed which model the performance of MULTISAFE during query processing. One equation is developed for each of the data dependent access conditions that are discussed in chapter III. Each of these equations shows the performance of MULTISAFE as a series of processing steps, without considering the concurrent nature of the modules in MULTISAFE. In chapter IX, the equations of chapter VIII are modified to reflect the ability of the MULTISAFE modules to work concurrently.

Chapter X contains several topics of interest. The two sets of equations from chapters VIII and IX are compared by analyzing one of the pairs of equations. The pair selected models the performance of MULTISAFE as it processes a query with a type A or B data dependency (see chapter III). A query with either of these types of data dependency is one of the more common types of data accessing queries. Next, query processing without any form of data protection is discussed. An equation is developed which models the performance of a protection-less MULTISAFE system as such a system processes a query. This equation is compared with the

equation from chapter IX that models the performance of MULTISAFE as it processes a query with a type A or B data dependency. The final topic presented in chapter X concerns an alternate assumption to one given in chapter VI. The assumption which is discussed and modified in this section concerns which of two MULTISAFE modules, working concurrently to process a query, finishes first.

Chapter XI ends this paper with a summary of the conclusions that are made throughout the paper. Some possible future work is discussed as well. The one appendix has a list of all of the variables which are used in the many equations developed in the paper.

Chapter II

A PREDICATE-BASED PROTECTION MODEL*

Hartson's predicate-based model of protection began as a semantic model for protection languages [HARTH75, HARTH76a, HARTH76b]. It has both an authorization process and an enforcement process. Protection requirements (authorizations) are presented to the authorization process. Subsequently, access requests are presented to the enforcement process which, by consulting the authorization information, renders an access decision.

An emphasis on authorization draws several other concepts into the model. For example, the "authorizer" emerges as an entity distinct from the "user." Authority can now be decentralized, if desired, from the bottleneck of a single database administrator. The model introduces a general dependency of the access decision on system state, rather than using the fixed relationship between users and data

---

*This chapter is taken entirely from [HARTH80]. It is adapted to an implementation model from the formal model of [HARTH76a]. The following notation will be used: Underscored upper case letters will denote sets while lower case letters, possibly subscripted, will denote set elements. Finally, subsets will be denoted by upper case letters, possibly subscripted. Subscripts will follow the associated variable, delineated by square brackets when necessary for clarity. For example, s, s[1], or s1 is an element of s, while s = {s1,s2} is a subset of $\underline{S}$={s1,s2,s3}.

afforded by the more conventional access matrix. Also, the concept of "ownership" acquires a role more general than its traditional one.

2.1 SETS AND PREDICATES

Several sets are involved in the model. An important set is U, the set of individual users, who will be making access requests. Sets of users will be called user groups. There is also a set of individuals who will be granting authorizations, the authorizers, A. An authorizer is an owner of data and has the responsibility of making authorizations, i.e., of determining who may share in the access of that data and in what manner. In the literature the term "user" is almost universally used to denote both users of the database system and users of the protection system. The distinction is emphasized here by using the term "authorizer."

The next set of the model is the one that contains data. D is the set of all data in the database. Real world access requests and data definitions deal with subsets of the database, as well as with individual data elements. Since this protection model is not tied to a particular model of data, the way in which elements and subsets of D are defined will be left unspecified until the specifics of the implementation are discussed in chapter V.

The operations a user may perform on the data also comprise a set, O. Examples of operations are: OPEN, CREATE, READ, WRITE, APPEND, UPDATE, DELETE, EXECUTE, COMPARE, RETRIEVE, OWN, and SUBOWN. The model is extensible in that it allows new data types and operations to be added.

A set can be described explicitly by enumeration of its members or implicitly by a predicate, or condition, used as its characteristic function. For example, let c(b,B) be a condition defining set B. Membership of any potential element b in set B is dependent on c(b,B) in the following way:

$$B = \{b \mid c(b,B) \text{ holds}\}$$

Since the truth value of c(b,B) can, in general, vary with time (with database system state), membership in B is a dynamic property.

If the set in question is U, a group of users, then c(u,U) determines whether or not an individual user u is a member of user group U at a specific instant in time and c(u,U) is called a user group defining condition. Similarly, c(d,D) is called a data defining condition, where d is a data element (such as a single record, tuple, or field value) and D is some subset of D.

Another important use of a predicate is an access condition, specifying the condition (involving the system state) under which a given type of access is to be allowed.

The set of all access conditions known to the system at any time will be denoted by C. Within this model there are no restrictions on what type of variables can be used in any of the conditions. However, in the implementation model which follows each condition type has been restricted to certain classes of variables without any loss of generality.

A user group can be defined by a list of user identifiers. The examples which follow illustrate a second way--by the use of predicates. The simplest of these definitions is the one for a standing group called GENERAL. Its defining predicate is:

c(u,GENERAL) : "true"

Thus, because this predicate has a constant "true" value, every user is implicitly and unconditionally a member of the GENERAL group, sharing its minimum level of privileges. This is an answer to the problem of wasting storage to represent, for example, the rights of everyone to use a public file [LAMPB71].

The following is typical of the user group defining conditions which can be dynamically evaluated.

c(u,u1) : dept (u) = 13 v project (u) = 'design'

When a user logs in, if he or she is either in department 13 or part of the design project, he or she becomes a member of user group U1 (until he or she logs off or until his or her

membership in U1 is tested again) and is given its access rights. (See the footnote in section 2.3 regarding symbols denoting logical operators in predicates.)

References to data, whether for access or for access control, are very dependent upon the data model and its method of data definition. In most real systems it will often be desirable to use a combination of explicit naming and implicit predicate-based definitions. The explicit naming, such as is done by providing relation and attribute names, describes a domain in which a data dependent predicate is to be applied. Following is an example which uses both the relation name "personnel" and content values for attributes named "dept" and "salary:"

c(d,D3) : RELATION (d) = 'personnel' & dept (d) = 7

& salary (d) < 20000

& (ATTRIBUTE(d) = 'name' v ATTRIBUTE(d) = 'address')

This data definition refers to a "fragment" of the PERSONNEL relation, restricted to tuples with a DEPT value of 7 and SALARY values of less than 20,000, projected to the NAME and ADDRESS attributes.

Access conditions--provisos which must be met before certain accesses are allowed--can be associated with global system variables such as the time-of-day clock, modes of operation, status indicators, flags, and internal codes.

Many of these indicators contain information known to the database system about the current transaction. As an example, suppose that certain accounting information can be entered only on Fridays. For 1981, this is represented by the following access condition:

$$C: \text{MOD}(day,7) = 2$$

where MOD is the modulo function and day is the Julian day number on the system calendar. Access conditions can also be data dependent predicates.

2.2 ACCESS REQUESTS AND AUTHORIZATIONS

The purpose of a database system is to serve the user in response to an access request. An access request, or query, is a triple:

$$q = (u,o,D)$$

where $u \in U$, $o \in O$, and $D \subseteq D$. This represents a request by an individual user u for a single operation o to data subset D. Of the three elements, the user actually provides only o and D. The user identifier, u, is supplied by the system in an "unforgeable" manner.

An instance of an authorization is a 5-tuple:

$$p = (a,U,O,D,c)$$

where $a \in A$, $U \subseteq U$, $o \in O$, $D \subseteq D$, and $c \in C$. This 5-tuple represents a declaration by authorizer a that each and every individual

user in user group U may do any or all of the operations in O to any or all of the data elements in D only if the access condition (predicate) c has a value of "true." For reasons of accountability, each authorization is marked with the identity of the authorizer who created it. Only that authorizer can modify or delete that authorization. However, a subowner (one to whom ownership has been explicitly granted by the owner who created the data) of a given subset of data, D, can issue other authorizations governing the access to D. Thus, several different authorizers can each create an authorization for the same user group, operation, and subset of the database, but perhaps with different access conditions. A subowner cannot grant OWNership to others.*

Consider this protection requirement, taken from [CONWR72b]: A user may see and update only "financial" parts of each record in a given personnel file, and only between 9 a.m. and 5 p.m. from a specific terminal in the payroll department. The user group is defined by:

c(u,U6) : dept(u) = 'payroll' & term1(u) = 'payoffice'

The data to be protected is:

c(d,D5) : RELATION(d) = 'personnel' &

---

*These ownership rules are a matter of high level policy, and other policies are possible.

(ATTRIBUTE(d) = 'salary' v ATTRIBUTE(d) = 'rate')

assuming the salary and rate attributes are the "financial" parts of the tuples. The access condition is:

c7: TIME > 0900 & TIME < 1700

The authorization which then ties these definitions together is:

p4 = (a,U6,{READ,WRITE},D5,c7)

2.3 AUTHORIZATION AND ENFORCEMENT

Before the authorization process translates the protection language expressions of authorizers into internal representations of access control information, it first validates the right of the authorizer to make the authorization. The process controls granting, as well as revocation, of rights; keeps lists of authorizer-created definitions; controls the display of access control information; and keeps a journal of all transactions with the protection system.

Before proceeding with the enforcement process, it is convenient to define some projection functions which operate on n-tuples. In general, where p = (x1,x2,...,xn), a set of projection functions i(p) = x[i], is defined for i = 1,2,...,n. Since the tuples of this model carry specific element names, mnemonic meaning is better served by the use

of the names of the elements instead of a general scheme with subscripts. Therefore, for p = (a1,U3,O7,D4,c6) : a(p)=a1, U(p)=U3, O(P)=O7, D(p)=D4, and c(p)=c6. Also, let $\underline{P}$ = {p1,p2,...,pk} be the net collection of valid authorizations received up to a given time. The enforcement process is now stated in the context of a request, q = (u,o,D), and the access decision is computed. (A similar set of projection functions is used on q: u(q), o(q), and D(q)). An example will be developed concurrently to illustrate the application of enforcement.

Assume that, at a given point in time, the following authorizations and the associated definitions exist in the system:

$\underline{P}$ = {p1=(a,U1,o,D2,"true"), p2=(a,U1,o,D3,c1),
p3=(a,U2,o,D1,c2), p4=(a,U2,o,D4,c3),
p5=(a,U3,o,D1,c4), p6=(a,U4,o,D1,c5),
p7=(a,U4,o,D2,c6), p8=(a,U4,o,D3,c7)}

Without loss of generality, a single authorizer a and a single operation o are assumed. The remaining information in $\underline{P}$ can be represented as the matrix of access conditions in Figure 1.

	D1	D2	D3	D4
U1	F	T	c1	F
U2	c2	F	F	c3
U3	c4	F	F	F
U4	c5	c6	c7	F

Figure 1. Matrix of Access Conditions

Now, consider a request $q = (u,o,D)$. The enforcement algorithm follows.

(1) Determine from the set of known user groups those groups to which the requesting user $u(q)$ belongs.

Example. This step is done by searching lists of user identifiers (explicit user group definitions) for $u(q)$ and by evaluating the predicates of the implicit user group definitions. Assume that it is determined that $u(q) \in U2$ and $u(q) \in U4$.

(2) Collect from $\underline{P}$, the set of all authorizations received to date, those authorizations which have the user groups determined in step (1) as elements. The result, called the franchise of the user, is given formally as:

$$F(u) = \{p \in \underline{P} \mid u(q) \in U(p)\}$$

Steps (1) and (2) can be done once per terminal session, at log-in time.

Example. For the example, this is {p3,p4,p6,p7,p8}, since these are the authorizations which mention U2 or U4.

(3) Determine from the set of known data subsets (already defined for purposes of authorization) those data subsets which have data in common with the data subset requested in q.

Example. This determination is dependent on the data model and its method of data definition. Suppose, for the

example, it is found that D(Q) is found to have elements in common with D1 and D3.

(4) Determine the set of authorizations that name data subsets found in step (3). Formally, this is denoted as:

$$\{p \in \underline{P} \mid D(q) \ \underline{XN} \ D(p) \neq \{\} \}.$$

Example. Here, the set which mentions D1 or D3 is {p2,p3,p5,p6,p8}.

(5) Determine the set of authorizations which specified o(q) as a data operation. Formally, this is denoted as:

$$\{p \in \underline{P} \mid o(q) \in O(p)\}.$$

Example. Here, this is all of $\underline{P}$, since only one operation is being considered.

(6) Determine those authorizations common to the sets found in steps (2), (4), and (5). As this is the set of authorizations which pertain to this specific request, it is called F(q), the <u>franchise</u> for the request q. Formally,

$$F(q) = \{p \in \underline{P} \mid u(q) \in U(p) \ \& \ D(q) \ \underline{XN} \ D(p) \neq \{\} \ \& \ o(q) \in O(p)\}$$

All of the enforcement process so far is summarized in this one expression. F(q) is the set of authorizations which are <u>applicable</u> to q (i.e., which participate in the access decision for q).

Example. In the example, the Franchise of q is {p3,p6,p8}.

(7) The set of data subsets named in the authorizations of F(q) is called D*(q), the "data reference" of q. Let D*(q) = {D1*, D2*, ..., D[d]*}. This step is to determine if the requested data subset D(q) is covered by the data reference D*(q); i.e., it must be true that D(q) $\subseteq$ UN D[i]*, $\forall i \in [1,d]$. That is, every element of D(q) must be contained in some member of D*(q).

Explanation. Having some overlap with authorized data is not enough for all of the requested data to be accessed. All of the requested data must be subject to an authorization. At this point, under a policy of "full enforcement" [HARTH77] (all data may be accessed or none is), failure of the covering check terminates the enforcement process with a flat rejection. Under a "partial enforcement" policy (those parts covered can be further considered for access), the non-covered parts are eliminated by setting:

$$D(q) = D(q) \text{ XN } D^*(q)$$

(8) Partition the set F(q) into equivalence classes based on the relation such that two authorizations are in the same class if and only if they specify the same data subset. Formally, this is accomplished as follows.

---

*Throughout the paper, the following notation is adopted:
UN 'set union'

Partition F(q) into d equivalence classes such that the i-th class is:

$$F[i](q) = \{p \in F(q) \mid D(p) = D[i]^*\}$$

$\forall i \in [1,d]$. Construct a temporary composite authorization for class, F[i](q):

$$p'[i] = (*, u(q), o(q), D[i]^*, \vee\{c(p[j]) \mid p[j] \in F[i](q)\})$$

where "v" denotes the logical OR of the set of access condition predicates, c(p[j]), over all j such that $p[j] \in F[i](q)$. The franchise is now the collection of all these composite authorizations:

$$F(q) = \{p'[i] \mid i = 1,2,\ldots,d\}$$

Example. The partitions are {p3,p6}, corresponding to D1, and {p8}, corresponding to D3. For the example, F(q) becomes {(*,u(q),o(q),D1,c2vc5), (*,u(q),o(q),D3,c7)}. The result of this step, for each data subset in D*(q), is an OR of the user's rights over all groups of which he or she is a member. This may be easier to see in the matrix of access conditions, Figure 2.

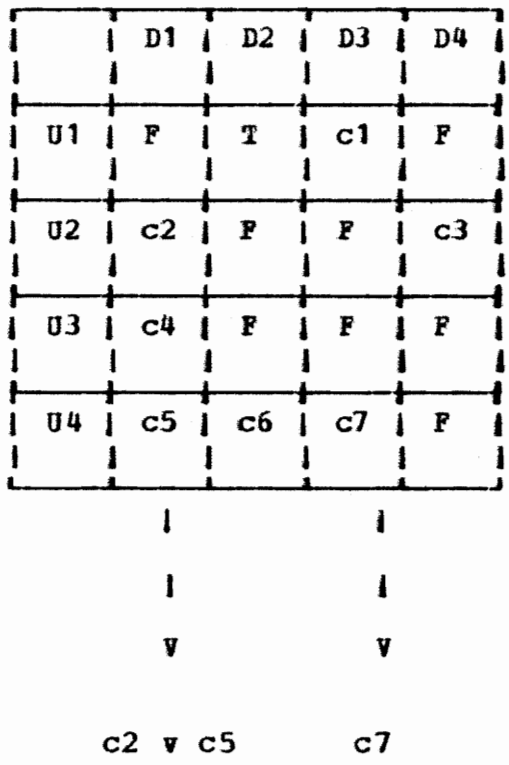


Figure 2. The OR Function Over User Groups

(9) Find EAC(q), the effective access condition corresponding to the request q, by performing the logical AND over the access conditions of the members F(u,q):

$$EAC(q) = \&(c(p'[i])), \text{ over } i=1,2,\ldots,d$$

Example. EAC(q) = (c2 v c5) & c7

(10) Evaluate the effective access condition and render an access decision: permit the requested access if EAC has a value of "true"; deny if otherwise.

The above sequence of ten steps, while useful for explaining the enforcement process, is not followed directly by the implementation. Some short cuts will be described in chapter V. Also, the relational model has turned out to be an ideal environment in which to implement this model. The collection of authorization tuples is a relation. Many of the steps of the enforcement algorithm are nothing more than relational calculus descriptions of queries over the authorization relation, operating on subsets of its tuples.

XN 'set intersection'
& 'logical AND operator'
v 'logical OR operator'
● 'OR over a bit-by-bit AND' (see section 5.3)
{} 'the null set'
∈ 'is a member of' (set membership)

In [HARTH76b] this basic model of protection is extended to history keeping (access decision dependency on previous database events) and auxiliary program invocation (procedures triggered by database events).

Chapter III

CLASSES OF ACCESS DECISION DEPENDENCY*

A predicate-based protection model has been described in [HARTH76a]. The generality introduced there by the use of predicates as conditions of access implies a variety of ways in which the access decision can be dependent on different kinds of system information. Three broad categories of access conditions which can be placed on a user of database are:

1. System dependent conditions.
2. Query dependent conditions.
3. Data dependent conditions.

3.1 SYSTEM DEPENDENT ACCESS CONDITIONS

An access condition (predicate) is a system dependent condition if its Boolean value can be ascertained from information available about the general system state. Such a condition might require that the time of day be between 8 a.m. and 5 p.m., allowing database operations only during regular working hours, or only on certain days of the week or month.

---

*This chapter is adapted from [HARTH81] and [BALLE81].

3.2 QUERY DEPENDENT ACCESS CONDITIONS

A condition is a query dependent condition if its Boolean value can be ascertained from the query itself. A SEQUEL-type [CHAMD76] language is used to illustrate. A simple SEQUEL request takes the form:

```
SELECT <attribute-list>
FROM   <relation-name>
WHERE  <selection-predicate>
```

A query dependent access condition can limit the relations upon which the user can operate, the attributes from which the user can SELECT, and the attributes which can be used in the WHERE clause.

The following database of three relations is used to illustrate the various kinds of dependency. (Much of this discussion of access decision dependency is taken from [BALLE81, HARTH80].)

EMP (EMP#, NAME, SALARY, BIRTH_YEAR, DEPT, YRS_SERVICE)
containing information about employees with attributes of employee number, name, annual salary rate, year of birth, department, and year of service

TAX (EMP#, NBR_DEPS, EARNED)
containing tax information about employees, with attributes of employee number, number of dependents, and year-to-date earnings

MGR (DEPT, MGR_NAME, STATUS)
with the manager names for each department and information on "clearance status"

Managers, being employees as well, are also listed in the EMP relation. EMP and TAX are separate relations for operational reasons and not, for example, because of third normal form requirements. The clearance status information in the MGR relation is provided by each manager for his or her department; the semantics are unimportant here.

As an example of a query dependent condition, consider an authorization which allows the selection of NAMEs from the EMP relation as long as the WHERE predicate does not specify SALARY values. Similarly, a selection of SALARY from the EMP relation may be disallowed if NAME also appears in the SELECT clause attribute list.

3.3 DATA DEPENDENT ACCESS CONDITIONS

A condition is a *data dependent condition* if its value cannot be ascertained without a retrieval (or perhaps several retrievals) from the database. The classes of data dependency are presented in approximate order of increasing complexity.

3.3.1 Class A Data Dependency

A condition has a *class A data dependency* if it has truth values in a one-to-one relationship with the tuples retrieved in response to a query, and the values depend on

the retrieved fields of the retrieved tuples. With such a dependency, an access decision must be made for each tuple retrieved. Consider the following simple, unqualified query, Q:

```
SELECT NAME, SALARY
FROM   EMP
```

For Q, an example of a class A access condition is:

SALARY <20,000

3.3.2 Class B Data Dependency

A condition has a class B data dependency if the condition has values in a one-to-one relationship with the tuples retrieved in response to a query, but the information needed to evaluate the condition is not present in the retrieved fields. Instead, it is in another field (or other fields) of the retrieved tuples. For Q, this access condition (which must be computed from BIRTH_YEAR) is class B:

age < 40

3.3.3 Class C Data Dependency

A condition has a class C data dependency if the condition has values in a one-to-one relationship to the elements retrieved in response to a query, but the information needed to evaluate the condition is not present in any fields of the relation. Instead, the information is available from another relation (or relations) in the database.

The following access condition is of class C with respect to query Q:

NBR_DEPS <=3

3.3.4 Class D Data Dependency

A condition has a class D data dependency if the condition has values in a one-to-n relationship with the tuples retrieved in response to a query, and the information needed to evaluate the condition requires only one database retrieval. This information may be:

1. in the response to the query,
2. in the non-response attributes of the response tuples, or
3. found by making a single additional request to the database, independent of the user's query.

For simplicity, in all cases, an extra access from the database is conservatively assumed to be needed. Given query Q, this access condition is of class D:

name of manager of finance department ≠ 'Joe'

An access to the MGR relation is needed. If the name (MGR_NAME) of the finance department manager is Joe, access is immediately denied to all tuples selected in response to Q. If the name of the finance department manager is not Joe, then access is allowed to all of the response tuples.

3.3.5 Class E Data Dependency

A condition has a class E data dependency if the condition involves aggregate information such as sums, counts, averages, or derived data. There are five subclasses of E which are detailed below. These subclasses approximately parallel the classes of non-aggregate dependencies which were described above.

3.3.5.1 Subclass Ea Data Dependency

A condition has a subclass Ea data dependency if the condition involves aggregate information in a one to n relationship with the tuples retrieved in response to a query, and the information depends on the retrieved fields of the retrieved tuples. Once all responses to a query have been retrieved, the values in the appropriate fields are used to compute the aggregate totals. The access condition can then be evaluated. For the query Q, an example of a class Ea access condition is:

AVG SALARY < 20,000

3.3.5.2 Subclass Eb Data Dependency

A condition has a subclass Eb data dependency if the condition involves aggregate information in a one to n relationship with tuples retrieved in response to a query, but the information needed to evaluate the condition is not present in the retrieved fields. Instead the information is present in another field or fields of the retrieved tuples. Once all responses to a query have been retrieved, the access condition can be evaluated. For Q, the following access condition is of class Eb:

AVG YRS_SERVICE > 10

3.3.5.3 Subclass Ec Data Dependency

A condition has a subclass Ec data dependency if the condition involves aggregate information in a one to n relationship with the tuples retrieved in response to a query, but the information needed to evaluate the condition is not present in any of the retrieved fields. Instead, the information is available from another relation or relations in the database. Given query Q, the following access condition is of class Ec:

SUM EARNED > 50,000

3.3.5.4 Subclass Ed Data Dependency

A condition has a subclass Ed data dependency if the condition involves aggregate information, in a one to one relationship with the tuples retrieved in response to a query, that depends upon one or more fields in the retrieved tuples. For query Q, this access condition is of class Ed:

```
AGE WHEN HIRED > 25
```

The age of each employee when hired must be computed from the BIRTH_YEAR and YRS_SERVICE fields, and from the known current date.

3.3.5.5 Subclass Ee Data Dependency

A condition has a subclass Ee data dependency if the condition involves aggregate information in a one to one relationship with the tuples retrieved in response to a query, but the information depends upon database retrievals in addition to those necessary to respond to the query, or independent from those necessary to respond to the query. Given the following unqualified query Q':

```
SELECT EMP#, EARNED
FROM   TAX
```

The following access condition is of class Ee with respect to query Q':

```
AGE WHEN HIRED > 25
```

For each tuple retrieved from the TAX relation, another tuple must be retrieved from the EMP relation in order to evaluate the access condition. The EMP# contained in the TAX tuple is used to retrieve the proper tuple from the EMP relation.

3.4 DEPENDENCY AND BINDING

The time at which each part of an access condition can be shown to be satisfied for a given query is called the binding time for that part. The time at which a complete access decision can be reached depends upon the type of dependencies within the access condition.

3.4.1 Binding of System Dependent Access Conditions

If a condition has only a system dependency, it can be evaluated as soon as the system state can be determined. A system dependent condition can be applied at any time from the logging in of the user (possibly even earlier), to the time the results of a service request are returned to the user. As an example, a policy may limit database access to times between 8 a.m. and 5 p.m. For most purposes that can be interpreted to mean that database operations may be allowed only if the request for service occurs in the prescribed time range. Or, if timing is critical, it may mean

that the response be returned to the user only within the same time interval. These two interpretations imply two different binding times, one at the start of the service and one at the end.

3.4.2 Binding of Query Dependent Access Conditions

If a condition has only a query dependency, then enforcement can be performed upon receipt of a query, leading to a relatively early binding time. If the query is:

```
SELECT   NAME, SALARY
FROM     EMP
WHERE    DEPARTMENT = 'finance'
```

and the access policy for this user and the EMP relation is that NAME is allowed as long as SALARY is not also requested, then the query can be immediately rejected.

A query dependent access condition may involve combinations of the SELECT attributes, relations used, and attributes in the WHERE clause. All forms of query dependent access conditions can be bound at query processing time.

3.4.3 Binding of Data Dependent Access Conditions

If a condition has a data dependency, the binding time depends upon the class of the dependency. Class A or B conditions (being one-to-one with retrieved tuples) must be bound repeatedly as the responses to a query return from the

database. A class C condition requires one additional database retrieval to be made for each response to a query. Thus, class C conditions require a later (and more costly) binding time than that of classes A and B. Upon completion of this retrieval, a decision can be made which affects only one response. A class D condition requires that one additional database retrieval be made for the entire query. Upon completion of this retrieval, a decision can be made which is in effect for all responses to the query. Therefore, although the class D binding time is very late, it is not as costly (because it is not as frequently needed) as that of class C conditions. A class E condition requires that aggregate information be computed before the condition can be evaluated.

For subclasses Ea, Eb, and Ec access conditions, as soon as all of the responses to a query have been received, and necessary computations have been performed, a decision can be made that affects all of the responses to the query. For a subclass Ed access condition, as soon as an individual tuple from the database has been obtained and necessary computations have been performed, a decision can be made which affects that tuple response only. For a subclass Ee access condition several database retrievals must be made for each tuple retrieved in response to a query, causing a delay

which results in a later binding time than for a subclass Ed access condition.

3.5 ENFORCEMENT AND DISCLOSURE POLICIES

Policy considerations at a high level are fundamental to the behavior of the system. Two enforcement resolution policies and two disclosure policies are considered. An *enforcement resolution policy*, introduced in [HARTH77], is a policy which determines how much data is to be returned to the requester when only part of the response to his or her query is accessible according to the authorization information. A *disclosure policy*, introduced here, is a policy which decides how informed the user should be of the authorizations that pertain to his or her database operations. Enforcement policy applies only to how the enforcement process is performed, whereas disclosure policy is used only in formulating user messages which explain the response or lack of response.

3.5.1 Definitions

If any part of any response to a query fails to meet the authorization requirements for access, a *full enforcement policy* requires that no information at all be passed to the user. Full enforcement is an 'all-or-nothing' policy.

In contrast, partial enforcement is a policy which permits the user to be passed all the individual responses to the query that pass all applicable access conditions. Only the responses that fail an access condition are withheld.

Associated with full and partial enforcement are two disclosure policies, complete and null disclosure. Complete disclosure is a policy under which a user is made aware of all the authorizations that control his or her database operations. Null disclosure is a policy whereby the user is kept completely unaware of the authorizations placed on his or her database activities.

In a database system none of the four possible policy combinations may be universally applicable. A different policy may be chosen depending upon the individual user and on the information that is being protected. Determining which of the four policies best suits a particular user over a given part of the database is generally left to the discretion of the authorizer.

If, under a full disclosure policy, the user is told he or she may get the NAME and ADDRESS of all employees, the user, nevertheless, is still not told about the existence of other information, possibly in the same relation. The user merely knows that there may or may not be other attributes of the relation that he or she cannot query. Full disclo-

sure can also lead to illegal inference of data values. For example, if the user asks for the SALARY of the employee named Jones and the access condition is to allow SALARY if SALARY < $20,000, the user upon being informed of the access condition that is violated will know that Jones earns more than $20,000. Further discussion of the inference problem is beyond the scope of this paper, but is treated extensively in the literature (as examples, see [DENND79, DOBKD79]).

3.5.2 Partial Enforcement

The INGRES database system is an interesting example of a partial enforcement policy, around which the entire protection system is built. The protection mechanism of INGRES [STONM74] is used to modify incoming queries so that the resulting query is guaranteed to request only authorized data. The modified request is then processed by the database retrieval programs without further need for enforcement of access rules. To greatly oversimplify (details are found in [STONM74]), data dependent access conditions are ANDed to the search qualification predicate (in the WHERE clause) of the incoming query. For example, if a given employee of the ABC Company has access only to records of Department A, his or her general request for information about the entire com-

pany is modified with a predicate such as: DEPT = 'A'. Thus, there is a response to the request, but it contains no information from tuples having a Department value other than A.

It is possible for this policy to cause difficulties if the requester does not understand that the response corresponds to a query other than the one he or she submitted. It is particularly serious if the requester asked for, say, the average salary in Company ABC. If the user was unaware that the result was an average for only Department A, he or she would unknowingly be dealing with false information. Of course, a full disclosure policy would provide the information necessary to know that the results are inaccurate and why they are inaccurate. On the other hand, a null disclosure policy may be suitable if a user does not need to be aware of some parts of the database. For example, if a user, whose view of the world includes only a certain department, asks for the average salary, he or she gets only the average salary of the employees that he or she has the authorization to access, and need not be concerned as to the restrictions placed on his or her request by an authorizer. This approach is exemplified by the early ASAP security measures, based on user views [CONWR72a].

3.5.3 Full Enforcement

In contrast, consider a full enforcement access policy which allows all employee salaries except the salary of a manager. If the user asks for the average salary of all employees, he or she is told that no response is allowed. Only a correct (from a global viewpoint) query, requesting the average salary of all employees except the manager, is allowed, and it produces a correct response.

Chapter IV

INTRODUCTION TO MULTISAFE*

A MULTIprocessor system for supporting Secure Authorization with Full Enforcement (MULTISAFE) for shared database management is being developed [TRUER80] by Trueblood at the University of South Carolina and Hartson at Virginia Tech. Performance improvements are expected to be achieved by a combination of multiprocessing, pipelining, and parallelism. The MULTISAFE protection processor can meet complex policy requirements with flexible, generalized protection mechanisms.

The system configuration is based on functional division into three major modules:

1. the user and application module (UAM)
2. the data storage and retrieval module (SRM)
3. the protection and security module (PSM)

Each module is implemented on one or more processors forming the multiprocessor system. In MULTISAFE all three modules function concurrently. The UAM coordinates and analyzes user requests at the same time that the SRM generates responses for requests. Simultaneously, the PSM continuously performs security checks on all activities. Figure 3 illustrates the logical relationships among the three modules.

---

*This chapter is taken from [HARTH80].

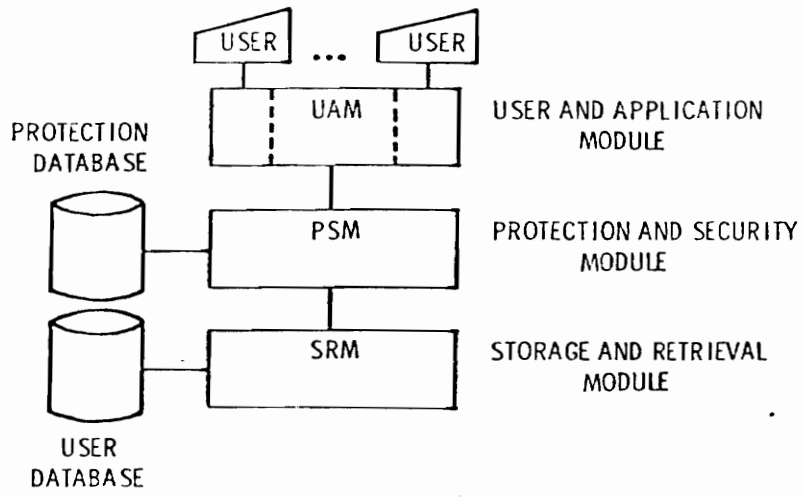


Figure 3. Logical Relationship Among the Three Modules

All processing within MULTISAFE is initiated and controlled by events occurring within the message flow, including such events as the transmission of data to and from the database.

Typically, the concepts of isolation and separation have been considered important for supporting data security. However, the protection question can still be considered to be open, because physical isolation is not a guarantee of security. It is at the logical level that evidence must be given that an architecture does indeed support data security. Unless communication among the system components can be shown to be logically secure (in terms of both message control and message content), security is not necessarily gained by isolation. Consequently, special attention is given to intermodule communication in [TRUER80].

In MULTISAFE, messages are sent between modules via encapsulated data types. Its contents are set and checked by protected procedures which are invoked parametrically. No user or user process can directly access these message objects. The primary mechanisms are structured and verifiable. For a single user, it is shown in [TRUER80] that the only message path between the UAM and the SRM is established by a sequence of carefully defined operations on abstract objects in the PSM. It is also shown that the message sequences from multiple users, introduced for the sake of

concurrency, can be effectively serialized, leaving intact the security of the single user case.

Messages from either authorizers or users are subject to two kinds of security checking: 1) checking specific to the request, and 2) system occupancy checking. System occupancy checks relate to overall permission to be an active user of the system, without regard to how the system is being used. The system occupancy check is always made in conjunction with log-in. For example, the conditions (separate from user identification) for a given system user may be that occupancy is allowed only between 8:00 a.m. and 5:00 p.m. System occupancy checking at data request time and other times provides (optional) additional binding times [HARTH77] for these conditions.

In this paper the concern is with the implementation of the authorization and enforcement mechanism within the PSM of MULTISAFE.

Chapter V

A DATABASE APPROACH TO PROTECTION IMPLEMENTATION*

5.1 THE RELATIONAL MAKEUP OF THE PROTECTION DATABASE

The database system used by the accessors of data is a relational DBMS, called the User Data Base (UDB) (See figure 3 in Chapter IV). The authorization information (called the Protection Data Base, or PDB) is stored in relational form, as is the authorization information of System R [GRIFP76]. The PDB uses a fixed set of relations and a specially modified version of the UDB for storage and retrieval. The fixed set of relations (initially in the system and, thereafter, never created or destroyed) in the PSM includes USERS and AUTHS. The USERS relation contains userid and account information of each individual user, as well as definitions for all user groups. The AUTHS relation contains the authorization information, i.e., who has the right to do what operations on what data and under what access conditions. In other words, AUTHS contains the set P of section 2.3. SCHEMAS, a fixed relation in the SRM, provides the database directory containing the definitions of the user-created data relations.

---

*This chapter is taken from [HARTH80].

5.1.1 Data Definitions

The database system directory is contained in the relation SCHEMAS. SCHEMAS conceptually consists of variable length tuples of the form SCHEMAS (RELATION, NBR_ATTRS,

NAME_OF_ATTR_NO_1, TYPE_OF_ATTR_NO_1,
ATTR_NO_1_INDEXED,
NAME_OF_ATTR_NO_2, TYPE_OF_ATTR_NO_2,
ATTR_NO_2_INDEXED, ...,
NAME_OF_ATTR_NO_N, TYPE_OF_ATTR_NO_N,
ATTR_NO_N_INDEXED).

This conceptual view of SCHEMAS is what the user sees. The implementation details, described in the following paragraph, are transparent to the user.

The underlying implementation of the relation SCHEMAS has been simplified by separating it into two parts. The Data Base System Directory (DBSD) contains the relation name (RELATION) and number of attributes (NBR_ATTRS) for each relation in the system. In addition, the DBSD contains a pointer to a Relation Directory (RD) which contains the other attributes of the SCHEMAS relation. There is one DBSD for the entire system and one RD for each data relation. In addition to these attributes from the SCHEMAS relation, the

RD contains a count of the number of tuples in the relation and pointers, into the data storage area and to secondary index B-trees, which are needed to retrieve the data tuples.

5.1.2 User and User Group Definitions

In MULTISAFE individual users are identified by the standard attributes: userid, account number, project name, password, etc. Users can also be grouped together to simplify the authorization of access to shared data. User groups are a generalization of the Multics (and others) "project" concept. The USERS relation is a 6-tuple of information about system users and groups of users with the form:

USERS (GROUP_NAME, USER_ID, ACCT_NO, TERM_NO, PROJ_NAME, PASSWORD).

In MULTISAFE, users can be grouped either explicitly by enumerating the members of the set or implicitly by stating a set-defining predicate. At log-in time, all such predicates are tested against the user's characteristics. In this particular implementation, user group defining predicates are restricted to the use of attributes found in the USERS Relation (e.g., user_id, account number, terminal number, and project name). The user becomes a member of each group for which the corresponding predicate is true. More discussion

of the use of these predicates is given in section 5.2. Implicit membership holds for the duration of the terminal session until logout, or until such time that the group defining conditions need to be re-evaluated.

Section 5.1.4 shows an example of a USERS relation. Several of the values in the tuples contain a "*". A value of "*" implies that this field of the tuple matches any value in the corresponding predicate of a login enforcement query (see section 5.2.1).

The tuple with GROUP_NAME GENERAL is a tuple which provides a user with a minimum set of access rights to the database. The existence of the GENERAL group implies that each legal user in the MULTISAFE system is a member of at least two groups: his or her own group, identified by the tuple that contains the user's login attributes, and the GENERAL group (see section 5.2.1). Notice that each of the fields in the GENERAL group tuple, except for the GROUP_NAME field, contains a "*". The definition of the "*" value implies that the GENERAL group tuple matches the login enforcement query issued for each potential MULTISAFE user.

5.1.3 Authorization Information

The AUTHS relation contains the set of all authorizations. Each row of AUTHS is a 6-tuple of information governing which users or groups of users have access to any of the system relations. The AUTHS relation has the form:

AUTHS (AUTHORIZER, GROUP_NAME, OPERATIONS,
RELATION, ATTRIBUTES, ACCESS_CONDITION).

AUTHS contains a dynamic record of the current state of authorization information as a result of the cumulative effect of all valid authorizations received up to a given point in time. In each tuple of AUTHS the attribute RELATION and the bit-coded attribute ATTRIBUTES denote a data relation or a projection of a data relation. The correspondence of the bits of ATTRIBUTES to attributes of the data relation is given by the order of the attributes as defined when the relation was created. A value of '1' in a bit position means that the corresponding attribute is included in the projection defined by that AUTHS tuple. A value of '0' means that it is not. Since the operation set is fixed, for convenience of implementation, the OPERATION attribute is bit-coded, too. To illustrate the semantics of AUTHS, assume that a data relation called EMP has been defined by a user named SMITH. Consider the contents of AUTHS shown below.

AUTHS:

	AUTHORIZER	GROUP_NAME	OPERATIONS	RELATION	ATTRIBUTES	ACCESS_CONDITION
1	SMITH	SMITH	OWN,RETRIEVE,UPDATE,...	EMP	111111	T
2	SMITH	GROUP1	RETRIEVE	EMP	010010	C1
3	SMITH	GROUP2	RETRIEVE	EMP	011000	C2

For the relation EMP, the ATTRIBUTES bits represent the attributes of EMP in the order they were defined:

EMP (EMP#, NAME, SALARY, BIRTH_YEAR, DEPT, YRS_SERVICE)

The first tuple of AUTHS denotes unconditional ownership (including full access rights) of EMP, the relation created by Smith. This tuple was entered into AUTHS as one result of Smith's command that created EMP. The OWN right is what allowed Smith to grant RETRIEVE rights to GROUP1 and GROUP2, represented by the second and third tuples in AUTHS. The second tuple defines a projection of the EMP relation containing only attributes NAME and DEPT; the third, a projection over NAME and SALARY. C1 and C2 are the names of the access conditions for the two grants. A value of "T" for ACCESS_CONDITION denotes an unconditionally "true" truth value in the first tuple of AUTHS.

To simplify checking of access conditions when retrieving data, the access conditions are stored in a separate relation

with the ACCESS_CONDITION attribute of AUTHS containing pointers into the CONDITIONS relation. The method of evaluation for a condition is determined by its dependency class (see section 3.3 and all of its subsections). The treatment of access conditions is further discussed in section 5.5.

Section 5.1.4 shows examples of AUTHS and CONDITIONS relations.

5.1.4 Examples of USERS, AUTHS, and CONDITIONS Relations

USERS:

	GROUP_ NAME	USER_ ID	ACCT_ NO	TERM_ NO	PROJ_ NAME	PASS WORD
1	SYSADMIN	SYSADMIN	0	*	SYS	ADMIN
2	GENERAL	*	*	*	*	*
3	FIKE	FIKE	120001	*	DESIGN	CHET
4	TALBOTT	TALBOTT	120004	*	IMPL	TOM
5	LUNDIN	LUNDIN	120003	42	IMPL	ROB
6	GROUP1	TALBOTT	*	*	*	PASS1
7	GROUP1	LUNDIN	*	*	*	PASS1
8	GROUP2	*	*	*	IMPL	PASS2

AUTHS:

	AUTHORIZER	GROUP_NAME	OPERATIONS	RELATION	ATTRIBUTES	ACCESS_CONDITION
1	--	SYSADMIN	OWN,INSERT,...	AUTHS	111111	T
2	--	SYSADMIN	OWN,INSERT,...	USERS	111111	T
3	--	SYSADMIN	OWN,INSERT,...	SCHEMAS	111...	T
4	SYSADMIN	GENERAL	INSERT	USERS	111111	C1
5	SYSADMIN	GENERAL	RETRIEVE	USERS	111110	T
6	SYSADMIN	GENERAL	INSERT,RETRIEVE,...	SCHEMAS	111...	T
7	SYSADMIN	GENERAL	RETRIEVE	AUTHS	111111	C2
8	SYSADMIN	GENERAL	UPDATE,DELETE	AUTHS	111111	C3
9	TALBOTT	TALBOTT	OWN,RETRIEVE,...	EMP	1111	T
10	TALBOTT	GROUP1	UPDATE	EMP	1010	C4
11	TALBOTT	GROUP2	RETRIEVE	EMP	1001	C5
12	TALBOTT	LUNDIN	UPDATE,DELETE	EMP	1000	C6

CONDITIONS:

COND_NAME	PREDICATE
C1	NEW (GROUP_NAME) ≠ NEW (USER_ID)
C2	(u(q) ∈ AUTHS.GROUP_NAME) OR (u(q) = AUTHS.AUTHORIZER)
C3	u(q) = AUTHS.AUTHORIZER
C4	DEPT = 'D1'
C5	DEPT IN ('D1','D2','D3')
C6	SALARY < 25000

5.2 LOGGING-IN TO MULTISAFE

5.2.1 Log-in Enforcement

In order for defined users to make use of the database, they must log into the system. When a user logs in, he or she provides certain information such as userid and password, and possibly account number and project name. The terminal number is determined by system supplied information. These log-in attributes are stored in a temporary relation, L, as shown in the following example:

L:

USER-ID	PASSWORD	TERMINAL
LUNDIN	ROB	17

The enforcement process must decide if the log-in is to be allowed. It does so by formulating this query*:

```
INTO LOG
RETRIEVE *
FROM USERS
WHERE USERS.USER_ID = L.USER_ID AND
      USERS.GROUP_NAME = L.USER_ID AND
      USERS.TERMINAL = L.TERMINAL AND
      USERS.PASSWORD = L.PASSWORD.
```

Such a tuple (or tuples) corresponds to the user and his or her password. If found, the log-in is allowed. Tuple 5 or USERS (see section 5.1.4) is found in this example.

---

*All names of temporary relations used by the enforcement process are actually qualified by the USER_ID, in order to keep them separate in a multiuser environment.

The enforcement process next determines those user groups of which this user is a member, by the following query. The computation of this set of user groups corresponds to step 1 of the enforcement algorithm given in section 2.3.

```
INTO UG
RETRIEVE GROUP_NAME, PASSWORD
FROM USERS
WHERE USERS.USER_ID = LOG.USER_ID AND
      USERS.ACCT_NO = LOG.ACCT_NO AND
      USERS.TERM_NO = LOG.TERM_NO AND
      USERS.PROJ_NAME = LOG.PROJ_NAME.
```

For the example USERS relation of section 5.1.4, user Lundin will match the following group names with the above query: LUNDIN, GROUP1, GROUP2, and GENERAL. (As mentioned earlier, a value of "*" in the relation will match any value in the query predicate). The user must provide the passwords for each of these groups, except for GENERAL, in order to use the rights of each one. A user can exclude himself or herself from a group for a terminal session by withholding its password.

After log-in, the authorization information pertinent to this user must be gathered together (into the user's "franchise") in order to make access decisions in response to access requests by the user. The creation of the user's franchise is discussed next.

5.2.2 Determining the User Franchise

After log-in, in anticipation of having to make access decisions on user requests, the enforcement process creates a temporary relation containing that authorization information from AUTHS which is pertinent to just this user and his or her user groups. This set of authorization tuples is called the user's franchise and its computation corresponds to step 2 of the enforcement algorithm given in section 2.3.

```
INTO FRANCHISE
RETRIEVE *
FROM AUTHS
WHERE AUTHS.GROUP_NAME IS IN UG.GROUP_NAME
```

Given that the user's franchise is non-empty, the requesting user is now legally occupying the system and has access rights to some parts of the database. FRANCHISE defines this user's access rights for this terminal session until logout, or until some change in authorization requires FRANCHISE to be computed, dynamically, again. Lundin's franchise contains tuples 4 through 8 and 10 through 12 of AUTHS (see section 5.1.4) by virtue of his membership in GENERAL, GROUP1, and GROUP2.

5.3 MAKING AN ACCESS DECISION

The enforcement process will be described in terms of single variable (single relation) queries. It has been shown [WONGE76] that all queries can be decomposed into a set of single variable queries. In SEQUEL, a nested query is an illustration of a multi-variable query. Assume the existence of a second data relation,

DEPART (DNO, MGR, FLOOR).

DEPART gives, for each department number, the name of its manager and the floor (of the building) on which it is located. A query which requests the names of all people who work on the seventh floor is:

```
SELECT NAME
FROM EMP
WHERE DEPT IS IN
    SELECT DNO
    FROM DEPART
    WHERE FLOOR = 7
```

The inner and outer parts of this nested query can each be treated as a single variable query, even though such a view may not reflect the way in which retrieval is implemented. Under a partial enforcement policy, each part would retrieve only those tuples passing the data dependent security checks for the corresponding relation.

In order to illustrate the enforcement process, suppose that Lundin enters this query into the UAM:

```
RETRIEVE NAME, SALARY
FROM EMP
WHERE SALARY < 13,000.
```

Initial parsing determines and stores the query attributes in a temporary relation, Q, as shown here:

Q:	OPERATION	RELATION	ATTRIBUTES	WHERE COND
	RETRIEVE	EMP	1010	SALARY < 13,000

The ATTRIBUTES value in Q denotes all attributes of EMP that appear anywhere in the query. In general, this refers to all attributes in RETRIEVE, UPDATE, or WHERE clauses. INSERT and DELETE commands, which affect whole tuples, imply that all attributes of the relation are involved.

Previous sections (5.2.1 and 5.2.2, respectively) have detailed steps 1 and 2 of the enforcement process (section 2.3). Next to be considered are the data subsets described in steps 3 and 4 of section 2.3. The use of a relational data model and the bit-coded ATTRIBUTES attribute simplifies these concepts. The data subsets of step 3 in the enforcement algorithm are those data relation projections defined in RELATION and ATTRIBUTES of FRANCHISE which overlap D(q), the requested data subset. D(q) is composed of Q.RELATION and Q.ATTRIBUTES, taken together. In section 2.3, overlap of the requested data, D(q), and the data defined in an

authorization, D(p), was determined by: $D(q) \underline{XN} D(p) \neq \{\}$. In the implementation model that overlap is determined by this predicate:

```
FRANCHISE.RELATION = Q.RELATION
                     AND FRANCHISE.ATTRIBUTES ⊛
                     Q.ATTRIBUTES
```

where ⊛ is the logical OR over the bits of a bit-by-bit AND over the bits of the two ATTRIBUTES values being compared. Specifically, if A and B are bit coded as (a1 a2 ... an) and (b1 b2 ... bn), then A ⊛ B = (a1 & b1) v (a2 & b2) v ... v (an & bn). Practically, A ⊛ B has a truth value of one ("true") if A and B have a value of one in common at any of the bit positions (i.e., if A and B have any overlap).

The computation of <u>the franchise of the query</u>, F(q), in step (6) of section 2.3, combines the results of steps (2), (4), and (5). In the implementation model, F(q) is computed as follows:

```
INTO FQ
RETRIEVE *
FROM FRANCHISE
WHERE Q.OPERATION IS IN FRANCHISE.OPERATION
    AND Q.RELATION = FRANCHISE.RELATION
    AND Q.ATTRIBUTES ⊛ FRANCHISE.ATTRIBUTES
```

The important attributes of FQ are shown below, for the example:

RELATION	ATTRI-BUTES	ACCESS_CONDITION
EMP	1010	C4
EMP	1001	C5
EMP	1000	C6

In the relational database implementation, again, this is a very simple operation. In the example, Lundin's request results in having FQ contain tuples 10, 11, and 12 of AUTHS (which, by this point, were in FRANCHISE per section 5.2.2). Formally, in step 6 of the enforcement algorithm, this computation is:

$$F(q) = \{p \in \underline{P} \mid u(q) \in U(p) \;\&\; D(q) \;\underline{XN}\; D(p) \neq \{\} \;\&\; o(q) \in O(p)\}$$

The characteristic function defining this set is a boolean predicate of three ANDed factors. The first, $u(q) \in U(p)$, was taken into account when FRANCHISE was computed from AUTHS (see section 5.2.2). This is computed at log-in rather than at query processing time, because of performance considerations. The second and third factors of F(q) are represented by the WHERE clause in the above query.

D*(q), the "data reference", and its covering of D(q), the requested data subset, are next to be considered. In the implementation model, D*(q) is simply the projection of FQ over RELATION and ATTRIBUTES. Since a partial enforce-

ment policy [HARTH77] will be implemented, the question of covering can be solved simply by setting

$$D(q) = D(q) \underline{XN} D^*(q)$$

as in step 7 of the enforcement algorithm (see section 2.3). Within the implementation described here, both D*(q)--all of its tuples--and D(q) refer to the same relation, namely Q.RELATION (which is EMP in the example). Therefore, the above equation is satisfied by projecting the user query response tuples over attributes allowed by D*(q). The allowed attributes are indicated by the presence of a "one" in the ATTRIBUTES of a D*(q) tuple. In other words, the set of allowed attributes is denoted by "ones" in an OR taken down each column of bits in the ATTRIBUTES values of FQ. In the example, the ATTRIBUTES values of the D*(q) tuples are 1010, 1001, and 1000. The bit-wise OR over these values yields 1011, meaning that only NAME, SALARY, and DEPT can be returned to Lundin for this query. Thus, the request, involving only attributes NAME and SALARY, is indeed covered by D*(q). This step enforces all query dependent conditions (see section 3.2).

In step (8) of the enforcement algorithm, F(q) is partitioned into equivalence classes based on elements of D*(q). These partitions are groups of tuples in F(q) having the same value for ATTRIBUTES. Each such group has its

ACCESS_CONDITIONs ORed together. In this example, each of the three tuples of F(q) is a class by itself.

The resulting predicates for classes are then ANDed together to form the Effective Access Condition (EAC) of step (9), which represents all system dependent and data dependent conditions (see sections 3.1 and 3.2). In the example, EAC = C4 & C5 & C6. The evaluation of the EAC is discussed in section 5.5.

5.4 ENFORCEMENT OF AUTHORIZATION

Enforcement of all access operations on USERS and SCHEMAS is via authorizations stored in AUTHS, exactly the same as for operations on data relations. In controlling access to itself, AUTHS is used in a slightly different way, involving OWNership. In particular, enforcement for authorization insertion (and update) depends not on Q (the request to make an authorization), but on the contents of the tuple (the authorization itself) to be inserted (or updated). As an example, consider the request to insert a tuple granting GROUP1 the right to update NAME and SALARY in EMP for records having DEPT = 'D1'. This request appears in Q as follows:

Q: OPERATION	RELATION	ATTRIBUTES	WHERE COND.
INSERT	AUTHS	*	*

The important dependency in computing FA, the franchise for an authorization, is on the new (incoming) tuple for AUTHS:

```
INTO FA
RETRIEVE *
FROM FRANCHISE
WHERE ('OWN' IS IN FRANCHISE.OPERATION
      OR 'SUBOWN' IS IN FRANCHISE.OPERATION)
      AND NEW(AUTHS.RELATION) = FRANCHISE.RELATION
```

The requesting authorizer can have OWN as an OPERATION in his or her FRANCHISE only by being the creator of the relation in question. He or she can have SUBOWN as an operation by being a subowner. A subowner is one to whom ownership is granted by the creating owner. In general, the SUBOWN operation for a subowner can have an access condition predicate attached, making subownership a dynamic attribute. Requests to RETRIEVE from AUTHS are dependent on the tuples affected and are primarily governed by tuple 7 of AUTHS (see section 5.1.4). DELETE (and UPDATE) operations on AUTHS involve some potentially complex questions about revocation policy [GRIFP76].

The USERS and SCHEMAS relations could have a "DEFINER" column for the USER_ID of the person who entered each tuple. This would be useful for supporting policies that allow deletion or modification only by the person who originally made the definitions. The database approach to protection allows such flexibility in responding to a broad range of

policies by changing the PDB structure and contents (and sometimes the queries posed against the PDB by the enforcement process), but not changing the basic mechanisms that operate on the PDB.

5.5 THE TREATMENT OF ACCESS CONDITIONS AS BOOLEAN FUNCTIONS

Each access condition will actually create up to three tuples in the CONDITIONS relation. One tuple will contain the access condition, as typed in by the authorizer. The second tuple will contain the part of the condition which can be evaluated before data retrieval and the third tuple contains the part which is data dependent. Simple predicates are of the form: <attribute_name> <arithmetic_comparison_operator> <value>. Predicates, simple predicates connected with logical operators, are parsed and stored as coded strings. Attribute names are given numerical codes that refer to the ordinal positions of the attributes (and their relations) in SCHEMAS. Certain system variables (e.g., the time clock) are given special attribute numbers. Arithmetic comparison operators ($=, \neq, <, >, \leq, \geq$) and logical operators (AND, OR) are given numerical codes, too. A value is stored as a variable length item with its length coded as part of the value. When needed, a predicate can be rapidly expanded into a very simple logic tree and evaluated.

Chapter VI

PRELIMINARIES TO PERFORMANCE EQUATIONS

MULTISAFE, as with any computer system of some size and complexity, has various devices available for data storage. While there is no attempt being made to describe the type of hardware devices which could comprise MULTISAFE, it will be assumed, for the purposes of this paper, that the various relations in the User database and the Protection database are stored on disks. It is possible that any relation in the User database and most of the relations in the Protection database have the potential to become quite large. Some of these relations may not be referenced at all for varying lengths of time. These are some of the reasons disks have been chosen as the storage media for the relations in the system. During the course of any discussions, it will be explicitly stated that a relation is stored in main (primary) memory if it is believed that such an assumption is valid.

With relations being stored on disks, MULTISAFE will be constantly needing to perform retrievals of tuples from disk. In current multiprogramming systems, during the time needed to retrieve a tuple from disk, many instructions related to the processing of already retrieved data can be

executed by a processing unit, using primary memory where applicable. It will be assumed in many of the sections that processing of one tuple can be completed before another tuple has been retrieved to be processed. In order that these processes can be performed simultaneously, it will be assumed that MULTISAFE uses a double buffering scheme to retrieve data [WEIDG77]. In this scheme, two buffers exist in primary memory, any one of which may be selected as a place for retrieved data to be stored. While the second buffer is being filled, the data in the first buffer can be processed. Upon completion of this processing, the first buffer becomes available. Once the second buffer has been filled, the data in it is processed while the first buffer is reused as a place to store new data being retrieved. This process continues until all data has been retrieved and processed. If two buffers are not enough for a particular task, it will be assumed that more buffers are available as needed.

6.1 PRELIMINARY INFORMATION CONCERNING THE EQUATIONS

A large number of performance equations are developed in succeeding chapters. They are given reference numbers enclosed in "()", such as "(7.2)". They show the time needed to complete different types of processing. Some of

the variables in the equations have subscripts which are enclosed in "[ ]", such as B[U]. The names chosen for variables and subscripts are intended to provide some mnemonic clues concerning their meanings. A complete list of variables with descriptions of their meanings can be found in the appendix.

Three major sets of equations are developed. The first set represents the time needed to process a user login. These equations eventually lead to one equation which models the entire login procedure. The second set models the protection enforcement processing time for queries. One equation is developed which shows the time needed to perform preliminary processing on a query. As will be seen, the steps of this process are the same for any given database query. Then, equations are developed which model the time to complete processing of a group of several queries. Each query in the group has one associated access condition which represents one of the data dependencies discussed in section 3.3 and all of its subsections. In addition, the queries are assumed to have one user-defined access condition, or predicate, which narrows down the list of tuples, retrieved from the user's requested relation, that are potential candidates to be returned to the user. These two assumptions are similar in nature to those in [BANEJ78]. Whether a

tuple is returned depends on the access decision made using the access condition. The times to perform preliminary processing and to complete processing of a query are combined to form an equation which shows the time to process a given query.

The last set of equations deweloped are concerned with the same set of user queries discussed in the previous paragraph. This time the focus is on enhancements which can take place in the performance of MULTISAFE. Because of the modular nature of MULTISAFE, certain processes can proceed simultaneously. Turnaround time for user queries is decreased as a result. While it is beyond the scope of this paper to describe every enhancement in performance which can exist in MULTISAFE, several are pointed out and their effect exhibited on the performance equations concerning the user queries which were previously described.

6.2 ASSUMPTIONS CONCERNING TUPLE RETRIEVAL

For each of the types of queries considered, the following assumptions will hold:

1. The attribute referenced in the predicate of a query is a secondary key into the relation being queried.
2. An index exists on that key. This index is a B-tree [COMED79], with x levels.

From [WIEDG77], the time R' to retrieve a block of data from disk given that the starting address of the block is known is:

$$R' = s+r+btt \qquad (6.1)^*$$

where s is the seek time needed by the disk read/write head to move to the proper track, r is the rotational latency time for the beginning of the desired block to rotate under the disk head, and btt is the block transfer time, that is, the time to transfer the block of data from the disk to primary memory.

Using equation (6.1), let

$$R = s+r+btt+P$$
$$= R'+P \qquad (6.2)$$

where P is the amount of CPU processing time needed to determine the address of a desired block of data. Assuming that a needed tuple is located in a block of data whose disk address can be determined, equation (6.2) is the time needed to retrieve one tuple from disk. This variable R will be used throughout the paper in various equations.

Finally, given the value R and the assumptions about an index with X levels on a secondary key, the time needed to retrieve one tuple from disk using a B-tree index is:

---

*See the appendix for a complete description of the performance variables.

$$T[1] = (1+x)\ (s+r+btt+P)$$

$$= (1+x)\,R \qquad (6.3)$$

This equation is taken from [WIEDG77]. Each progression through a level of the B-tree index requires one disk access. When the last level has been retrieved, one more access is needed to get the desired tuple. The address of the tuple to be retrieved is determined from information contained in the last level of the B-tree.

The assumption that related tuples are physically clustered on disk [ASTRM76] will not be made. If n tuples are to be retrieved in response to a query, it will be assumed that n disk accesses are needed to get them, if only one index has been followed. Of course, the information stored in the bottom level of the index, which resides in primary memory once all preceding levels have been passed through, can be used to retrieve each of the n tuples. Thus, the time needed to retrieve n tuples from disk using the one index is:

$$T[n] = (n+x)\,R \qquad (6.4)$$

Several of the performance equations developed in succeeding sections reflect the fact that more than one relation has been referenced. Each corresponding B-tree index may have a different number of levels. In addition, the time R to retrieve one tuple from disk may be different for

various relations because of differences in the corresponding variables which make up equations 6.2 and 6.3. In these equations the corresponding variables x and R will be subscripted to differentiate them.

Chapter VII

PERFORMANCE EQUATIONS FOR USER LOGIN

In the sections below, one equation will be developed which indicates the time it takes for MULTISAFE to process a login request from a user. The login steps described in previous sections will be summarized below. Various assumptions will be discussed as well.

The Protection and Security module, or PSM, processes all login requests. It accesses information stored in relations which are located in the Protection database, or PDB. The assumptions detailed in the previous chapter regarding the use of B-tree indices to access tuples apply to any PDB relations accessed by the PSM.

The following is a list of variables which are used in this chapter:

j -- the number of user profile tuples retrieved from the USERS relation.

k -- the number of user group tuples retrieved from the USERS relation.

m -- the number of authorization tuples retrieved from the AUTHS relation.

P -- a quantity of CPU processing time.

p -- the probability of an event.

R -- the time to retrieve a tuple from a relation.

x1 -- the number of levels in a B-tree index into the USERS relation.

x2 -- the number of levels in another B-tree index into the USERS relation.

x3 -- the number of levels in a B-tree index into the AUTHS relation.

7.1 PERFORMANCE EQUATIONS FOR A SUCCESSFUL LOGIN

Performance equations for a typical successful user login into the MULTISAFE system will now be derived. The assumptions which were made in section 6.2 about retrieval of tuples and relations in general apply in this discussion. Several temporary relations used to store intermediate results acquired during login processing will be assumed to reside in primary memory. Thus, any tuples retrieved from these relations will not involve any disk accesses.

The steps involved in processing a login request, as detailed in chapter V of this paper and in [HARTH80] are summarized below. Some intermediate equations are developed as well.

1. MULTISAFE identifies itself to the user attempting to log in and requests user profile parameters. The user enters a set of parameters, which are stored in a temporary relation called L. The relation L is assumed to reside in primary

memory. MULTISAFE enters the user's terminal number into L. Some processing time P is used to create L and to store the parameters.

2. Using the parameters stored in L, the PSM generates a query which requests retrieval of one or more user profile tuples from the USERS relation. An amount of processing time P is consumed in generating and parsing this query. Since USERS resides in the PDB, some disk accesses are needed to move through the index and retrieve the desired tuples from USERS. Assume that the B-tree index used here has x1 levels. The PSM must perform one disk access for each user profile tuple retrieved from USERS. Assume that j such tuples are retrieved. Information in the tuples retrieved from USERS is stored in a temporary primary memory resident relation called LOG.

Note that the equation developed at this point begins where the parameters in step 1 are stored in the newly created relation L. The time needed by the user to type his or her userid and password is not part of this equation. Steps 1 and 2 lead to the following equation:

$$C[\text{profile}] = P + (j + x1)\, R[\text{USERS}] \qquad (7.1)$$

3. The PSM generates a query to the USERS relation which retrieves group names and passwords into UG of all user groups to which the logging in user belongs. Some processing time P is needed to generate this query. The PSM must perform several disk accesses to move through the appropriate B-tree index. This index may be a different index than the one used in step 2. As a result, it may not contain the same number of levels. Assume, then, that there are x2 levels in it. Once the last level of the index has been retrieved, (based on assumptions outlined in the previous chapter) the PSM must perform one disk access for every desired tuple to be retrieved. Assume that k user group tuples are retrieved from USERS. As above, the time needed by a user to type group passwords is not included in the equation below. Step 3 leads to the following performance equation:

$$C[\text{usergps}] = P + (k+x2)\,R[\text{USERS}] \qquad (7.2)$$

4. The final step in the process is the creation of this user's FRANCHISE relation. All tuples in the AUTHS relation which reference any of the user groups in UG are retrieved and placed in FRANCHISE.

Assume that m such tuples are retrieved. Similar to equation (7.2), there will be some processing time P needed to generate a query, plus (m+x3) disk accesses to retrieve the m tuples from AUTHS, where x3 is the number of levels in the B-tree index to the desired tuples. The equation for this step is thus:

$$C[auths] = P + (m+x3)R[AUTHS] \qquad (7.3)$$

The performance equation which represents the time needed to process a successful login is found by linearly combining equations (7.1) through (7.3). Since CPU processing times are not being broken down in detail, the values of P in each of the above three equations are combined into one overall value P in the equation below. This process of combining values of P will be done several other times in this paper as well.

The following performance equation for the time to process a successful login is produced:

$$\begin{aligned} C[slogin] &= C[profile] + C[usergps] + C[auths] \\ &= P + (j+x1)R[USERS] + (k+x2)R[USERS] \\ &\quad + (m+x3)R[AUTHS] \\ &= P + (j+k+x1+x2)R[USERS] \\ &\quad + (m+x3)R[AUTHS] \end{aligned} \qquad (7.4)$$

7.2 PERFORMANCE EQUATIONS FOR AN UNSUCCESSFUL LOGIN

There are several places during the MULTISAFE login procedure where a user login could be aborted. These possibilities and any resulting changes to the equations developed in the previous section will be considered below.

1. Login could fail for a user if an incorrect user ID, password, account number, and/or project number was supplied. The login failure occurs when an attempt is made to retrieve user profile tuples from the users relation. No such tuples will be found. Depending on administrative policy, the user might be given a few more chances to log in, in the event of a typing error, for example. Extra login attempts and the time to process them will not be considered here. Once the last level of the index being used has been retrieved, it can be determined that there is no tuple with parameters matching those supplied by the user. Thus there will be only x1 disk accesses performed instead of j*x1 as in equation 7.1. Otherwise there are no further alterations needed in 7.1. The amount of CPU processing time will undoubtedly be different, but since no breakdown of CPU time is being considered this difference will not be

reflected in 7.5. This discussion leads to the following equation:

$$C[\text{flogin1}] = P + (x1)\,R[\text{USERS}] \qquad (7.5)$$

2. If an individual user profile tuple is located, the USERS relation is again queried for group numbers of user groups to which the user belongs. Recall from the discussion in section 5.1.2 that a legal user is a member of at least two groups. These groups are the group represented by the individual profile tuples returned in the previous step, and a general group whose members have a set of minimal access rights to data in the database. Thus the request for group numbers returns at least the number of the general group, plus perhaps the numbers of other groups which contain the user as a member. Login will not fail at this point, which implies that equation 7.2 remains unchanged.
3. A final step for a potential login failure can occur if no tuples are returned from the AUTHS relation for storage in the FRANCHISE relation. Instead of (m+x3) disk accesses, only x3 accesses are needed to determine that no AUTHS tuples should be returned. Equation 7.3 is changed as described above to:

$$C[noauths] = P + (x3)\,R[AUTHS] \qquad (7.6)$$

The complete equation at this step is formed by combining 7.6 with the equations 7.5 and 7.2 which give the time needed to process steps that are completed before this step. The following equation is produced:

$$\begin{aligned} C[flogin2] &= C[profile] + C[usergps] + C[noauths] \\ &= P + (x1)\,R[USERS] + (k+x2)\,R[USERS] \\ &\quad + (x3)\,R[AUTHS] \\ &= P + (k+x1+x2)\,R[USERS] \\ &\quad + (x3)\,R[AUTHS] \end{aligned} \qquad (7.7)$$

Notice that once again the CPU processing times in the three equations which were added together have been combined into one variable P.

7.3 TOTAL LOGIN PERFORMANCE EQUATION

One equation will now be developed which describes the login procedure in detail by using the equations developed in the previous sections.

An equation for a successful login was developed. Then two places in the login procedure where login could fail were identified. In the latter two cases equations describ-

ing the login procedure up through the points of failure were developed. These three cases are assumed to make up a list of all possible outcomes of an attempted login. As such, this list of outcomes is mutually exclusive and exhaustive. Each outcome, or event, has a certain probability of occurring. The events are defined as follows:

1. S = event of a successful login, as described by equation 7.4.
2. F1 = event that a user supplies incorrect user parameters, as described by equation 7.5.
3. F2 = event that no tuples are returned from the AUTHS relation, as described by equation 7.7.

The probabilities associated with these events are defined as follows:

1. PR(S) = p[s].
2. PR(F1) = p[f1].
3. PR(F2) = p[f2].

The fact that the three events are mutually exclusive and exhaustive implies that p[s]+p[f1]+p[f2] = 1.

The equation which totally describes the login procedure is found by multiplying the equations which describe each of the possible outcomes of an attempted login by their respective probabilities.

$$
\begin{aligned}
C[\text{tlogin}] &= p[s]C[\text{slogin}] + p[f1]C[\text{flogin1}] \\
&\quad + p[f2]C[\text{flogin2}] \\
&= p[s]\{P + (j+k+x1+x2)R[\text{USERS}] \\
&\quad + (m+x3)R[\text{AUTHS}]\} \\
&\quad + p[f1]\{P + (x1)R[\text{USERS}]\} \\
&\quad + p[f2]\{P + (k+x1+x2)R[\text{USERS}] \\
&\quad + (x3)R[\text{AUTHS}]\} \\
&= P + p[s](j+k+x1+x2)R[\text{USERS}] \\
&\quad + p[s](m+x3)R[\text{AUTHS}] \\
&\quad + p[f1](x1)R[\text{USERS}] \\
&\quad + p[f2](k+x1+x2)R[\text{USERS}] \\
&\quad + p(f2)(x3)R[\text{AUTHS}] \\
&= P + R[\text{USERS}]\{p[s](j+k+x1+x2) + p[f1](x1) \\
&\quad + p[f2](k+x1+x2)\} \\
&\quad + R[\text{AUTHS}]\{p[s](m+x3) + p(f2)(x3)\} \\
&= P + R[\text{USERS}]\{jp[s] + (p[s]+p[f2])(k+x1+x2) \\
&\quad + p[f1](x1)\} + R[\text{AUTHS}]\{mp[s] \\
&\quad + (x3)(p[s]+p[f2])\} \qquad (7.8)
\end{aligned}
$$

7.4 EXAMPLE

Using calculations from section 7.3, assign the following values to the variables in equation 7.8:

processing time:	P	= 1 ms
tuple retrieval time:	R[USERS]	= 70 ms
tuples retrieved from USERS:	k	= 5 tuples
tuples retrieved from USERS:	j	= 2 tuples
levels in an index into USERS:	x1	= 3 levels
levels in an index into USERS:	x2	= 4 levels
tuple retrieval time:	R[AUTHS]	= 60 ms
tuples retrieved from AUTHS:	m	= 10 tuples
levels in an index into AUTHS:	x3	= 5 levels
probability of a successful login:	p[s]	= 0.80
probability of incorrect user parameters:	p[f1]	= 0.19
probability of no authorization tuples:	p[f2]	= 0.01

These values, although arbitrary, were selected with the aid of [BANEJ78], [DONOJ72], [HAMAV79], and [WIEDG77]. The time needed to completely process a login request via equation 7.8 with the above values is:

$$C[tlogin] = P + R[USERS]\{jp[s] + (p[s]+p[f2])(k+x1+x2) + p[f1](x1)\} + R[AUTHS]\{mp[s] + (x3)(p[s]+p[f2])\}$$

$$= 1 + 70\{2(0.80) + (0.80+0.01)(5+3+4) + 0.19(3)\} + 60\{10(0.80) + 5(0.80+0.01)\}$$

$$= 1 + 70\{1.6+9.72+0.57\} + 60\{8+4.05\}$$

$$= 1 + 70(11.89) + 60(12.05)$$

$$= 1 + 832.3 + 723$$

$$= 1556.3 \text{ ms}$$

Chapter VIII

PERFORMANCE EQUATIONS FOR QUERY PROCESSING

In this chapter performance equations which are equivalent to the time needed for query processing in MULTISAFE will be developed. First, equations will be developed which show the time it takes to process queries up to the point at which any access conditions must be evaluated. No matter what query is being processed, the steps up to access condition evaluation are the same. Then the various types of data dependencies, which were discussed in sections 3.3.1 through 3.3.5.5, will be considered, and performance equations developed which show the time needed to process queries with simple access conditions which are representative of the data dependencies.

Finally, places in the process where there is a probability of partial or full enforcement of access conditions taking place will be considered. As a result of partial enforcement a user may be limited in some manner to the amount of information he or she is allowed to view in response to a query. As a result of full enforcement a user may have his or her query rejected.

8.1 PRELIMINARY QUERY PROCESSING

In the next few sections equations will be developed which describe the preliminary processing done for each query submitted by a user. The discussion will lead to one equation which totally describes this process. This equation will take into account possible points in the process where a query could be rejected.

The following variables are used in this section:

g -- the number of tuples retrieved from the CONDITIONS relation.

P -- a quantity of CPU processing time.

p -- the probability of an event.

R -- the time to retrieve a tuple from a relation.

x -- the number of levels in a B-tree index into the CONDITIONS relation.

8.1.1 Equations for Preliminary Processing

The steps below are a summary of the preliminary query processing steps undertaken by MULTISAFE. The material in this section is taken from [HARTH76a] and [HARTH80]. For a more complete discussion of these steps, the reader should consult these two references.

1. Once a user has submitted a query it is analyzed and parsed. Query attributes are stored in a

temporary relation called Q.

2. MULTISAFE generates a query which creates the franchise of the query. The user's FRANCHISE relation, which was created when the user logged in, and the Q relation are used to create this franchise relation, FQ. Since it is being assumed that both FRANCHISE and Q are stored in primary memory, no disk accesses are required to compute FQ.

3. The data reference D*(q) and its covering of D(q), the requested data subset, are next to be computed. The ATTRIBUTES field of the FQ relation is used to compute these items. As FQ is currently in primary memory, only a quantity of processing time P is used here. This step enforces all query dependent access conditions.

4. MULTISAFE then computes the Effective Access Condition for the query. FQ is partitioned into equivalence classes based on elements of D*(q). Each class has its access conditions ORed together. The resulting predicates for the classes are then ANDed together to form the EAC. Once again, only a quantity of processing time P is used in this step.

5. Now that the EAC has been formed, using the ACCESS_CONDITION pointers stored in FQ, the actual predicates which are addressed by the FQ ACCESS_CONDITION pointers must be retrieved from the CONDITIONS relation, which is stored in the PDB. Assume that there exists a B-tree index into CONDITIONS which has x levels, and that g predicate tuples are retrieved using this index.

From steps 1 through 5, equation 8.1 is produced:

$$C[EAC] = P + (g+x)\,R[CONDITIONS] \qquad (8.1)$$

Equation 8.1 represents the steps necessary to complete the preliminary processing of any user request. From this point on, in order to finish processing a user query, associated access conditions must be evaluated and tuples from the requested relation must be retrieved. Before the steps needed to do these things are considered, the equation developed above for preliminary processing will be completed by discussing points where partial or full enforcement of preliminary access conditions takes place. At some of these points, the potential exists for a query to be denied.

8.1.2 The Effects of Enforcement Policies

During the preliminary processing of a query, there are several points where the evaluation of a query could end because of a full enforcement policy being applied, or where certain restrictions may be imposed on information the user is eventually allowed to view because of a partial enforcement policy taking effect. The effects of the application of one or more full enforcement policies are of greater concern here. As stated, the effect of enforcing such a policy may be to immediately halt further processing of the query and deny it completely. The probabilities of a query successfully passing through such potential denial points during processing must be considered. Application of partial enforcement policies will not cause query denial, but may further limit the amount of information the user is allowed to view.

Referring to the preliminary query processing steps outlined in the previous section, once a user's query is parsed (step 1), in step 2 an attempt is made to create the franchise of the query. The user's FRANCHISE relation and the Q relation, in which the attributes of the parsed query currently reside, are used to create the FQ relation. If there are no tuples from FRANCHISE that are retrieved into

FQ, then a full enforcement policy must immediately be applied that denies the user's query. Why this policy is applied becomes clear by considering as an example the case of a user attempting to retrieve tuples from an EMPLOYEE relation. If there are not any tuples in FRANCHISE which refer to retrieval operations from EMPLOYEE, then the user is not allowed to look at any information in EMPLOYEE. If there are some tuples in FRANCHISE which refer to retrieval operations from EMPLOYEE, but the bit code in the ATTRIBUTES field of the Q relation does not have any overlap with any of the bit codes in these FRANCHISE tuples, then the user is not allowed to look at the fields of EMPLOYEE which are referred to in the Q.ATTRIBUTES bit code.

Based on the above discussion, query processing could potentially come to a halt after FQ is created. Let p1 be the probability that the query passes this point of enforcement and proceeds onward for further processing.

In step 3 of the previous section, the data reference and the requested data subset are computed. In this step, the attributes, or fields, of the requested relation that the user is allowed to see are determined. Each tuple in FQ contains a bit code that defines the attributes in the requested relation that are addressed by the access condition predicate in the tuple. By computing the OR down the

columns of bits in FQ.ATTRIBUTES the entire list of allowed attributes is determined. Under an "all-or-nothing" full enforcement policy, if this list of attributes did not completely cover the attributes that are reflected in the user's query (as indicated by the bit code in Q.ATTRIBUTES), then the query would immediately be denied. In section 5.3, however, it is stated that a partial enforcement policy is being implemented which will allow the user access to those attributes of the requested relation that are covered by D*(q). Thus, while the user may not be allowed to view all of his or her requested attributes, the user will at least get to see some of the attributes (assuming that the query passes future enforcement checks).

One final point to consider concerning the allowed attributes is the possibility that, after the OR of the columns of FQ.ATTRIBUTES bit codes is computed, the resulting bit code does not cover any of the user's requested attributes. If this condition could occur, then there is a possibility that the user's query would have to be denied. It becomes clear that this condition cannot occur by examining once again the process by which the (franchise of the query) relation FQ was formed. For a tuple from the user's FRANCHISE relation to be included in FQ, one of the conditions that it had to satisfy was that its bit code had to

have some overlap with the bit code in Q.ATTRIBUTES. Thus, each tuple in FQ covers at least one of the user's requested attributes, which implies that the bit code formed from the OR of the FQ.ATTRIBUTES bit codes covers at least one of the requested attributes.

Therefore, since a partial enforcement policy is in effect, the above discussion shows that at least one, and possibly all, of the user's requested attributes are allowed by the data reference covering. There is no chance for query denial here, and so the user's query passes to the next step.

The effective access condition, or EAC, is computed in step 4 of the previous section. In this step, it is decided exactly how the various individual access conditions will be combined to form the EAC. Actually, only the pointers in FQ.ACCESS_CONDITION are manipulated. Once the EAC is created, the actual access conditions are retrieved from the CONDITIONS relation.

Now begins the process of evaluating the access conditions. Some conditions depend on information stored in the database. These data dependent conditions will be considered later in the paper. Other system dependent conditions can be evaluated before tuple retrieval is begun. There is the possibility that after evaluating any such conditions

the query may be denied. If one such access condition, for example, is that the current time of day must be between 8 A.M. and 5 P.M. before the user can access a data relation, and the user has submitted the query at 7 P.M., then the query can immediately be denied, if evaluation of this condition gives the EAC a value of false. If this condition, known to be false for this query, is ORed with a data dependent condition, then it cannot be determined whether to deny this query at this time. Nevertheless, there is a chance that the query could be denied. Assume that the query successfully passes this point with probability p2.

8.1.3 Total Equation for Query Processing

Using the probabilities discussed in the previous section, one equation will now be developed which shows the time needed to complete preliminary processing. A summary of the steps discussed above is now presented.

1. The query is parsed, and an attempt is made to create the franchise of the query. The probability of a query continuing to the next processing phase is p1.
2. The data reference covering of allowed attributes is computed. There may be a partial enforcement policy applied which limits the amount of information the user eventually will see, but the query

does pass through this step with no chance of denial.

3. The effective access condition is created. All relevant access conditions are retrieved from the CONDITIONS relation. No enforcement policies are applied during this processing phase, so the query is passed to the next step.

4. Any system dependent conditions which are part of the EAC are evaluated. Depending on the contents of the EAC and the current state of the MULTISAFE system, it is possible that the query could now be denied. The probability that the query successfully passes this phase of processing is p2.

The following equation is derived using equation 8.1 and the probabilities p1 and p2:

$$C[qproc] = P + p1(P+(g+x)R[CONDITIONS] + p2(C[qcompn])) \quad (8.2)$$

The first P includes the primary memory processing time needed to parse the query and create the franchise of the query. The second P includes the processing time to compute the allowed attributes of the query and the EAC, to perform primary memory calculations needed during retrieval of CONDITIONS tuples, and to evaluate any system dependent access conditions. The expression designated C[qcompn] in

8.2 is the time to complete the processing of a query. Thus, upon substituting an equation for C[qcompn], C[qproc] becomes the time to completely process a query. C[qcompn] comes into play only if all steps of preliminary processing have been successfully passed by the query. Several different queries will be discussed in the next sections, and expressions for C[qcompn] derived for each. Sequence numbers will be substituted for the "n" in "qcompn" to distinguish between different values of C[qcompn].

8.1.4 Example

For the variables in equation 8.2., going from left to right through the equation make the following assignments:

processing time:	P	= 0.5 ms
probability query passes first enforcement phase:	p1	= 0.9
processing time:	P	= 1 ms
number of tuples retrieved from CONDITIONS:	g	= 5 tuples
number of levels in a CONDITIONS index:	x	= 3 levels
tuple retrieval time from CONDITIONS:	R[CONDITIONS]	= 55 ms
probability query passes second enforcement phase:	p2	= 0.8

These values, although arbitrary, were selected with the aid of [BANEJ78], [DONOJ72], [HAMAV79], and [WIEDG77].

Substitution of these values into equation 8.2 produces the following:

$$C[qproc] = P + p1(P + (g+x)\,R[CONDITIONS] + p2\,(C[qcompn]))$$

$$= 0.5 + 0.9(1 + (5+3)(55) + 0.8(C[qcompn]))$$

$$= 0.5 + 0.9(1 + 440 + 0.8C[qcompn])$$

$$= 0.5 + 0.9(441 + 0.8C[qcompn])$$

$$= 0.5 + 396.9 + 0.7C[qcompn]$$

$$= 397.4 + 0.7C[qcompn] \text{ ms}$$

The value to be substituted for C[qcompn] depends of course on the query being considered. According to the figures in this example, there is about a 70% chance that a query will make it through all of the preliminary checks; that is, about 30% of all queries will be rejected at some point during preliminary processing. These percentages may or may not seem unrealistic; however, the reader should note that this and other examples are for illustrative purposes only.

8.2 REMAINING EQUATIONS FOR QUERY PROCESSING

Now that preliminary processing has been completed, the pertinent data dependent access conditions, that depend on information stored in the database, can be applied to the user's query. One query for each of the data dependencies is considered. For this discussion, each query has one applicable access condition, stored in the FRANCHISE relation of the user making the request. This access condition is one of the types of data dependent conditions. Each query is assumed to have a WHERE clause with one (user-defined) predicate. Using the predicate, the SRM can select an appropriate B-tree index, via the proper relation directory (RD), to access the relation being requested in the user's FROM clause. Doing so narrows down the number of tuples to be retrieved to a number which is less than or equal to the cardinality of the requested relation. Without an applicable access condition, all of these tuples would be returned to the user. As it is, each tuple is just a candidate to be returned. Whether each tuple is returned to the user depends on the outcome of the access decision made by the PSM using the applicable access condition.

The following variables are used in this section:

d -- the number of levels in the DBSD B-tree index.

M -- the time for a memory to memory transfer of a tuple.

P -- a quantity of CPU processing time.

p -- the probability of an event.

R -- the time to retrieve a tuple from a relation.

R1 -- the time to retrieve a tuple from another relation.

R[DBSD] -- the time to retrieve a tuple from the DBSD.

R[RD] -- the time to retrieve a tuple from an RD.

y -- the number of tuples retrieved from a relation.

y' -- the number of retrieved tuples which are returned to the user.

z -- the number of levels in a B-tree index.

z1 -- the number of levels in another B-tree index.

8.2.1 Class A and B Dependencies

An access condition to be applied to a query has a class A or B data dependency if the condition has values in a one to one relationship with the tuples retrieved in response to a query, as stated in sections 3.3.1 and 3.3.2. The information needed to evaluate the condition is present in one or more of the domains, or fields, of the tuples being retrieved. For a class A condition these fields are also named in the SELECT clause of the query, while in a class B condition they are not. As such, the steps needed to process either type of query are essentially the same, with perhaps some slightly different CPU steps.

The first step in retrieving tuples in response to any query is to locate the tuple in the Data Base System Directory (DBSD) that contains information about the queried data relation. It is assumed that there is a B-tree index with d levels through which the DBSD can be accessed. Once the appropriate tuple is found, the proper relation directory (RD) can be accessed by using the relation directory pointers field in the DBSD tuple. From the RD, the SRM can locate the proper B-tree index to use to access the queried data relation. Assume that the RD being accessed has a tuple retrieval time of R[RD].

As each tuple is fetched from disk by the SRM it can immediately be decided whether to return the tuple to the user by evaluating the value of the domain referenced in the condition with the condition value itself. The tuple is simply accepted or rejected for return to the user depending on the outcome of this evaluation. Assume that y tuples are retrieved from the queried relation, and that of these y tuples y' successfully pass the condition and are returned to the user. Thus there will be y' memory to memory transfers of tuples, where each tuple is transferred from the SRM's primary memory to the primary memory of the UAM. To get the y tuples, assume that an index with z levels is used, and that the requested relation has a tuple retrieval time of R. Some projections may occur before information is passed to the user. For example, with a class B dependency the domain referenced by the access condition is masked out. The following equation is derived:

$$C[qcomp1] = P + (1+d)R[DBSD] + R[RD] + (y+z)R + y'm \qquad (8.3)$$

The part of the equation which reads "(1+d)R[DBSD] + R[RD]", plus a part of the CPU processing time P, represents the time needed for the SRM to make its way through the DBSD and RD associated with the queried data relation.

8.2.2 Class C Dependency

As described in section 3.3.3, a condition has a class C data dependency if it has values in a one to one relationship with the tuples retrieved in response to a query. However, the information needed to evaluate the condition must be obtained from another relation. Thus, when a tuple is retrieved from the queried or first relation a class C condition cannot be immediately evaluated because the information needed is not present in any of the values just returned. One or more of the returned values which can be employed as a key into the second relation must be used. From this second relation a tuple is brought back which will contain the information needed to evaluate the condition and render an access decision concerning the original tuple.

To process a query with a class C condition, two tuples must be retrieved from the DESD. Assume that a d-level index exists into the DBSD, and that the tuple retrieval time from the DBSD is R[DBSD]. After the two DBSD tuples have been retrieved, the associated RD's can be accessed. Assume that the RD used to get at the first relation has a tuple retrieval time of R[RD], while the RD used to access the second relation has a tuple retrieval time of R[RD1].

Assume that the first relation has a tuple retrieval time of R and an index to it with z levels. The second relation is assumed to have a retrieval time of R1 and an index to it with z1 levels. From the first relation it is assumed that y tuples are returned. Since one tuple is retrieved from the second relation for each tuple from the first, y tuples are retrieved from the second one as well. Assume that y' of the y tuples retrieved from the first relation satisfy the access condition and are returned to the user. The equation derived is as follows:

$$C[qcomp2] = P + (2+d)R[DBSD] + R[RD] + R[RD1] + (y+z)R + (y+z1)R1 + y'm \qquad (8.4)$$

8.2.3 Class D Dependency

A condition has a class D data dependency if it has values in a one to n relationship with the tuples retrieved in response to a query (section 3.3.4). The information needed to evaluate the condition can be obtained by retrieval of a single tuple from a relation.

Assume that a d-level index exists into the DBSD, which has a tuple retrieval time of R[DBSD]. Two tuples are retrieved from the DBSD, and then the two RD's are accessed. Assume that the RD which points to the queried relation has a retrieval time of R[RD], and that the RD which points to

the relation used to evaluate the access condition has a retrieval time of R[RD1].

Assume that there are z1 levels in the index used to go to the relation which contains the necessary information to evaluate the access condition. This relation is assumed to have a tuple retrieval time of R1. Upon evaluation of the condition using the retrieved information, either all the requested tuples in the relation being queried are retrieved and returned to the user, or none of them are. Assume that the condition is satisfied with probability p[s]. If it is satisfied assume that y tuples are retrieved from the relation being queried using an index with z levels. On the other hand, there is a probability of 1-p[s] that the user's query will be denied at this point in its evaluation, meaning that the user-requested relation is not accessed.

The following equation is derived:

$$C[qcomp3] = P + (2+d)R[DBSD] + R[RD] + R[RD1] + (1+z1)R1 + p[s]((y+z)R + yM) \qquad (8.5)$$

8.2.4 Class Ea and Eb Dependencies

A condition has a class Ea or Eb data dependency if the condition involves aggregate information in a one to n relationship with the responses to a query (sections 3.3.5.1 and 3.3.5.2). The field, or fields, containing this information

are named in the user's SELECT clause for a class Ea dependency. For a class Eb dependency these fields are not named in the SELECT clause. As in section 8.2.1, the steps to process either type of query are essentially the same.

Once again, assume that a d-level index exists into the DBSD, which has a retrieval time of R[DBSD]. One tuple is retrieved from the DBSD. This tuple is used to access the proper RD having a retrieval time of R[RD]. All tuples which could potentially be returned to the user must be retrieved. Assume that there are y of them, obtained via an index with z levels from a relation with a tuple retrieval time of R. When the y tuples have been retrieved, the processing necessary to evaluate the aggregate access condition is completed.

Assume that the condition is satisfied with probability p. If so, the y tuples are returned to the user. There is also a probability of 1-p that no tuples are returned to the user because the condition was not satisfied. The following equation is arrived at:

$$C[qcomp4] = P + (1+d)\,R[DBSD] + R[RD] + (y+z)\,R + p(yM) \qquad (8.6)$$

8.2.5 Class Ec Dependency

A condition has a class Ec dependency if it involves aggregate information that must be accumulated from a relation other than the one being queried by the user (section 3.3.5.3). This information is in a one to n relationship with the responses to the query. There will be some information obtained from the second relation for each tuple retrieved from the relation being queried by using the value, or values, from a tuple that is a key into the second relation. The information so retrieved from the second relation is used to accumulate the necessary aggregate total.

Assume that the DBSD has a tuple retrieval time of R[DBSD], and a d-level index associated with it that is used to retrieve two tuples. Using the first DBSD tuple, assume that the RD that is accessed is associated with the queried, or first, relation, and has a retrieval time of R[RD]. The second DBSD tuple is used to access the RD associated with the second relation. Assume that this RD has retrieval time R[RD1]. Assume that the first, or queried, relation has a tuple retrieval time of R, and an index with z levels used to retrieve y tuples. The second relation is assumed to have a tuple retrieval time of R1, and an index with z1 lev-

els. Since one access is made to this relation for each tuple retrieved from the first relation, there are y tuples obtained from this relation as well.

When tuple retrieval has been completed, the accumulated aggregate total(s) can now be used to evaluate the access condition on the query. It is assumed that the condition is satisfied with probability p, in which case the y tuples from the first relation are sent to the user. There is, of course, a probability of 1-p that the condition is not satisfied, in which case none of the first set of y tuples are sent.

Equation 8.7 results from this discussion:

$$C[qcomp5] = P + (2+d)R[DBSD] + R[RD] + R[RD1] + (y+z)R + (y+z1)R1 + pyM \qquad (8.7)$$

8.2.6 Class Ed Dependency

A condition has a class Ed data dependency if it involves aggregate information in a one to one relationship with the responses to the query (section 3.3.5.4). The aggregate information is present in one or more fields of the retrieved tuples.

Assume that the DBSD has a d-level index associated with it, and a tuple retrieval time of R[DBSD]. One tuple is retrieved from the DBSD. This tuple is used to access

the RD associated with the queried relation, which is assumed to have a retrieval time of R[RD]. For each tuple returned from the relation being queried, values from the proper fields are used to compute aggregate information which will enable an access decision to be rendered immediately. Assume that the relation, with tuple retrieval time R, has a z-level index used to retrieve y tuples. Of these y tuples, y' satisfy the access condition and are returned to the user. The following equation is derived:

$$C[qcomp6] = P + (1+d)R[DBSD] + R[RD] + (y+z)R + y'M \qquad (8.8)$$

8.2.7 Class Ee Dependency

An access condition has a class Ee data dependency if it involves aggregate information, in a one to one relationship with the responses to a query, that must be retrieved from a second relation (section 3.3.5.5). For each tuple returned from the first relation, the values from the proper fields which can be employed as a key into the second relation must be selected. The information returned from this relation is then used to compute the aggregate values needed to render an access decision concerning the first tuple returned.

As always, assume that the DBSD has a d-level index associated with it, and a tuple retrieval time of R[DBSD]. Two DBSD tuples are retrieved. The first one is used to access the RD associated with the first, or queried, relation. This RD has a retrieval time of R[RD]. The second DBSD tuple is used to access the RD, having a retrieval time of R[RD1], that is associated with the second relation. Assume that the first relation has a tuple retrieval time of R and a z-level index, while the second relation has a tuple retrieval time of R1 and a z1-level index. From the first relation assume that y tuples are retrieved. The discussion above dictates that y tuples are also retrieved from the second relation. Of the set of y tuples from the first relation, assume that y' of them satisfy the access condition and are returned to the user. Equation 8.9 results from these steps:

$$C[qcomp7] = P + (2+d)R[DBSD] + R[RD] + R[RD1] + (y+z)R + (y+z1)R1 + y'M \qquad (8.9)$$

8.3 EXAMPLE

In section 8.1.3, the equation (8.2) for C[qproc] was developed. This equation yields the time to completely process a query. The example in section 8.1.4 had values substituted for all the variables in C[qproc] except C[qcompn].

C[qcompn] is the time to complete processing for a given query. In this example, the equation developed for a query with a type A or B data dependency is used. This is equation 8.3, labeled C[qcomp1].

For the variables in 8.3, make the following substitutions:

processing time:	P	= 0.6 ms
number of levels in the DESD index:	d	= 3 levels
tuple retrieval time from the DBSD:	R[DBSD]	= 60 ms
tuple retrieval time from the RD:	R[RD]	= 65 ms
number of tuples retrieved from the queried relation:	y	= 50 tuples
number of levels in an index into the queried relation:	z	= 5 levels
tuple retrieval time from the queried relation:	R	= 75 ms
number of retrieved tuples returned to the user:	y'	= 25 tuples
time for a memory to memory transfer of a tuple:	M	= 2 ms

For C[qcomp1], these values produce:

$$C[qcomp1] = P + (1+d)R[DBSD] + R[RD] + (y+z)R + y'M$$

$$= 0.6 + (1+3)(60) + 65 + (50+5)(75)$$

$$+ (25)(2)$$

$$= 0.6 + 240 + 65 + 4125 + 50$$

$$= 4480.6 \text{ ms}$$

Substitution of C[qcomp1] for the variable C[qcompn] in equation8.2, and using the intermediate result in section 8.1.4, produces the following time to process a query with a type A or B dependency:

$$C[qproc] = 397.4 + (0.7)(C[qcomp1])$$

$$= 397.4 + (0.7)(4480.6)$$

$$= 397.4 + 3136.4$$

$$= 3533.8 \text{ ms}$$

Chapter IX

ENHANCEMENTS

One of the intents in the design of MULTISAFE was to optimize the turnaround time of responses to user queries by allowing some operations of query processing to be performed simultaneously. In the sections that follow, some logical enhancements are described which will help in realizing this goal. At the same time the effects on the equations, developed in the last chapter, that these enhancements produce are detailed. Much of the discussion in section 9.1 will apply in succeeding sections since in this section various concepts are discussed in greater detail.

The following variables are used in this chapter:

d -- the number of levels in a B-tree index into the DBSD.

g -- the number of tuples retrieved from the CONDITIONS relation.

M -- the time for a memory to memory transfer of a tuple.

P -- a quantity of CPU processing time.

p -- the probability of an event.

R -- the time to retrieve a tuple from a relation.

R1 -- the time to retrieve a tuple from another relation.

R[DBSD] -- the time to retrieve a tuple from the DBSD.

R[RD] -- the time to retrieve a tuple from an RD.

x -- the number of levels in a B-tree index into the CONDITIONS relation.

y -- the number of tuples retrieved from a relation.

z -- the number of levels in a B-tree index.

z1 -- the number of levels in another B-tree index.

9.1 CLASS A AND B DEPENDENCIES

When preliminary processing of a query with a class A or B data dependency is scheduled to begin, the query can be simultaneously transmitted to the SRM as it is sent to the PSM. Thus, while the PSM is engaged in making preliminary access decisions, the SRM can begin to retrieve tuples from the data base which are members of the relation being queried. Assume that a parsed version of the original query was transmitted to these two modules, and that each module has a copy of the data definitions (schemas).

Since many CPU steps can be performed in the amount of time it takes to perform one disk access, it will be assumed in this discussion that the tuple retrieval being carried out by the SRM will not be completed before the preliminary processing being done by the PSM. This assumption is supported by noting that in the MULTISAFE environment poten-

tially very large relations in the database are being queried by users. These relations quite probably are larger in terms of cardinality than the relations in the protection database being queried by the PSM. Thus by implication it is being assumed that more tuples are being retrieved by the SRM than by the PSM; that is, the SRM initiates more disk accesses than the PSM.

Assume that the query has passed all preliminary processing steps. In this discussion let p[p] be the probability that all the preliminary steps were passed. In the amount of time it has taken for the PSM to complete its processing, the SRM will have been able to retrieve, say, t tuples. The SRM will (possibly) be engaged in another tuple retrieval which has yet to be completed. Assume that some fraction of R time units have been used so far by the SRM in its current retrieval, say, R/r, where r is a random value such that $0 < 1/r <= 1$.

As the SRM continues to retrieve tuples the PSM can begin to evaluate the tuples already stored in SRM buffers. Some of these tuples may be returned to the user if they pass the test described in the query access condition, some may be rejected if they fail the test. At any rate, because of the relative speeds of the PSM evaluations versus the SRM tuple retrievals, the PSM will soon "catch up" with the SRM

to the point that once a tuple is retrieved and stored in an SRM buffer, the PSM can process this tuple completely before another tuple is returned to the SRM. This complete processing of each tuple, exclusive of the last tuple retrieved by the SRM, includes having the tuple sent to a UAM buffer via a memory to memory transfer if appropriate. Thus the entire process of query processing, assuming it is successfully carried through all the steps of preliminary processing, can be viewed in terms of the amount of time taken to retrieve all the tuples from the relation being queried. Recall that with the availability of double buffering techniques [WIEDG77] this retrieval of tuples can be viewed as a steady unbroken stream of disk accesses. The only other variables to consider are the processing time needed to evaluate the last tuple returned, and the processing time needed to parse the query and send copies of it simultaneously to the PSM and SRM. Once evaluation of the last tuple is complete there exists some probability p[t] that it must be transmitted to the UAM.

As has been the procedure in earlier sections, let R[DBSD] be the tuple retrieval time of the Data Base System Directory, which has a d-level B-tree index. One tuple is retrieved from the DBSD, which contains the information needed for the SRM to access the proper relation directory,

assumed to have a tuple retrieval time of R[RD]. The information in the RD leads the SRM to the B-tree which is used to access the queried data relation. Let y be the number of tuples retrieved using a z-level index into a relation with average tuple retrieval time R.

Before deriving an equation, the discussion of the previous paragraphs is summarized below.

1. The query is parsed, and copies are simultaneously sent to the SRM and PSM. This takes some CPU processing time.
2. Preliminary processing is completed with the probability being p[p] that the query successfully passed through all steps. Meanwhile the SRM retrieves all tuples which might be passed to the user. The PSM processes these tuples as they arrive from the database.
3. When the last tuple is retrieved, there is a probability of p[t] that it must be transmitted to the user.

These steps lead to the following equation:

$$C[qcomp1]' = P + p[p]((1+d)R[DBSD] + R[RD] + (y+z)R + P + p[t]M) \qquad (9.1)$$

To complete this equation, the possibility must be considered that the query will be rejected at some point during

preliminary processing. Referring back to equation 8.2, let C[qprelim]' be equal to C[qproc] without the term "p2(C[qcompn])". Thus C[qprelim]' is equal to the following equation:

$$C[qprelim]' = P + p1(P + (g+x)R[CONDITIONS]) \qquad (9.2)$$

There is a probability of 1-p[p] that a query is rejected at some point of preliminary processing. If this happens then the steps of query processing are no longer viewed in terms of tuple retrievals from the data base. Instead they are viewed in terms of equation 9.2. The complete equation for query processing for class A and B dependencies is thus:

$$\begin{aligned} C[qproc]' &= C[qcomp1]' + (1-p[p])C[qprelim]' \\ &= P + p[p]((1+d)R[DBSD] + R[RD] \\ &\quad + (y+z)R + P + p[t]M) \\ &\quad + (1-p[p])(P + p1(P + (g+x)R[CONDITIONS])) \end{aligned} \qquad (9.3)$$

9.2 CLASS C DEPENDENCY

In this section, as well as in succeeding sections concerning the other types of dependencies, the discussion in the last section about the concurrent functions of the MULTISAFE modules apply. Equations developed here and in following sections are developed in much the same manner as was equation 9.3.

Let R[DBSD], R[RD], and R[RD1] be the tuple retrieval times of the DBSD, which has a d-level index, and the two RD's selected from the DBSD information. Let y be the number of tuples retrieved using a z-level index from the relation queried by the user. From the relation where access condition information is located, y tuples are also retrieved but this time with a z1-level index. The two relations have tuple retrieval times of R and R1 respectively. The resulting equations are:

$$C[qproc]' = C[qcomp2]' + (1-p[p])C[qprelim]' \qquad (9.4)$$

$$C[qcomp2]' = P + p[p]((2+d)R[DBSD] + R[RD] + R[RD1] + (y+z)R + (y+z1)R1 + P + p[t]M) \qquad (9.5)$$

9.3 CLASS D DEPENDENCY

For this dependency class an access to another relation is needed. The information returned is used to evaluate the access condition. This access can be executed by the SRM first, while the PSM begins its preliminary query processing. Then the SRM can begin to retrieve tuples from the relation being queried. Once the PSM has completed its processing, while tuple retrieval by the SRM is continuing, it can use the information retrieved by the first SRM tuple retrieval to render an access decision. The only choices

possible here are to either return all applicable tuples to the user or deny the user access to any of the tuples. As in section 8.2.3 it will be assumed that the condition is satisfied with probability p[s]. If so, then the SRM continues to retrieve tuples which are immediately sent to the user via the UAM. As for the last tuple retrieved, there is no processing decision to make and no probability involved concerning whether to send it to the user. It is simply sent as soon as it is retrieved.

If the PSM denys the user's query based on the information present in the first tuple retrieved by the SRM, tuple retrieval is discontinued and no further processing is required. A denial takes place with probability 1-p[s].

The equations produced are as follows:

$$C[qproc]' = C[qcomp3]' + (1-p[p])C[qprelim]' \qquad (9.6)$$

$$\begin{aligned} C[qcomp3]' = P &+ p[p]((2+d)R[DBSD] + R[RD] \\ &+ R[RD1] + (1+z1)R1 \\ &+ p[s]((y+z)R + M) \\ &+ (1-p[s])C[qprelim]') \end{aligned} \qquad (9.7)$$

9.4 CLASS EA AND EB DEPENDENCIES

Processing for these two classes begins as with preceding classes, with the PSM and SRM simultaneously working on

their respective tasks concerning the query. Once the PSM finishes it cannot begin to render access decisions because all tuples addressed by the query must be retrieved in order to accumulate the necessary aggregate information. Of course, this information can be continuously updated as each tuple is returned to the SRM. Finally, when all tuples of the relation have been returned and the aggregate totals computed, the PSM renders an access decision concerning the tuples. All tuples will be returned to the user with probability p[s] if the condition is satisfied. Of course, the probability of no tuples being returned is 1-p[s].

$$C[qproc]' = C[qcomp4]' + (1-p[p])C[qprelim]' \qquad (9.8)$$

$$C[qcomp4]' = P + p[p]((1+d)R[DBSD] + R[RD] + (y+z)R + P + p[s]yM) \qquad (9.9)$$

9.5 CLASS EC DEPENDENCY

For this class of data dependency a second tuple must be retrieved for each tuple retrieved from the relation being queried. Information stored in the set of second tuples is used to compute an aggregate total. Thus the PSM does not render an access decision until all pairs of tuples have been retrieved by the SRM. Again it is assumed that the access condition is satisfied with probability p[s] such that the y tuples retrieved from the relation being queried are sent to the user. The condition fails, and no tuples are returned to the user, with probability 1-p[s].

$$C[qproc]' = C[qcomp5]' + (1-p[p])C[qprelim]' \qquad (9.10)$$

$$C[qcomp5]' = P + p[p]((2+d)R[DBSD] + R[RD] + R[RD1] + (y+z)R + (y+z1)R1 + P + p[s]yM) \qquad (9.11)$$

9.6 CLASS ED DEPENDENCY

In this class of dependency tuples are processed on an individual basis. Aggregate totals are computed for each tuple from information returned in one or more fields of the tuples. Then an access decision is rendered and the tuple is either transmitted to the user with probability p[t] or not transmitted with probability 1-p[t]. All tuples

returned except the last one can be processed while continuous retrieval of tuples is being carried out by the SRM.

$$C[qproc]' = C[qcomp6]' + (1-p[p])C[qprelim]' \quad (9.12)$$

$$C[qcomp6]' = P + p[p]((1+d)R[DBSD] + R[RD] + (y+z)R + P + p[t]M) \quad (9.13)$$

9.7 CLASS EE DEPENDENCY

Processing for this class of dependency is similar to the Ed class, except that as with several other classes the value (or values) which can be used as a key into a second relation is used to retrieve information needed to render an access decision concerning the corresponding first tuple. Each tuple is processed on an individual basis in this manner. The time needed to compute an aggregate total and render a decision only appears in the equation for the last pair of tuples returned by the SRM.

$$C[qproc]' = C[qcomp7]' + (1-p[p])C[qprelim]' \quad (9.14)$$

$$C[qcomp7]' = P + p[p]((2+d)R[DBSD] + R[RD] + R[RD1] + (y+z)R + (y+z1)R1 + P + p[t]M) \quad (9.15)$$

9.8 EXAMPLE

In section 8.1.4, parameters were used to compute a value for C[qproc]. C[qprelim]' (equation 9.2) was defined

using C[qproc]. Upon examination of equations 9.2 and 8.2 and the calculations carried out in section 8.1.4, it will become apparent that

C[qprelim]' = 397.4 ms

It was mentioned in section 8.1.4 that given the computed figure, there was a probability of 70% that a given query would pass through all of the preliminary checkpoints. In section 9.1, p[p] was defined as the probability that a query passes all preliminary checks. Therefore, in this example

p[p] = 0.7

For this example, equation 9.1 is used. This equation gives the time to complete processing of a query with a type A or B dependency assuming all preliminary steps were passed. Variables in this equation which correspond to those in equation 8.3 are given equivalent values as were assigned in section 8.3. For equation 9.1, make the following assignments:

processing time:	P = 0.6 ms
number of levels in the DBSD index:	d = 3 levels
tuple retrieval time from the DBSD:	R[DBSD] = 60 ms
tuple retrieval time from the RD:	R[RD] = 65 ms
number of tuples retrieved from the queried relation:	y = 50 tuples
number of levels in an index into the queried relation:	z = 5 levels
tuple retrieval time from the queried relation:	R = 75 ms

processing time:	P	= 0.001 ms
probability that last retrieved tuple is sent to the user:	p[t]	= 0.5
time for a memory to memory transfer of a tuple:	M	= 2 ms

Then:

$$C[qcomp1]' = P + p[p]((1+d)R[DBSD] + R[RD] + (y+z)R + P + p[t]M)$$

$$= 0.6 + 0.7((1+3)60 + 65 + (50+5)75 + 0.001 + 0.5(2))$$

$$= 0.6 + 0.7(240 + 65 + 4125 + 0.001 + 1)$$

$$= 0.6 + 0.7(4431)$$

$$= 0.6 + 3101.7$$

$$= 3102.3 \text{ ms}$$

Finally, from equation 9.3, the time to completely process a query with a type A or B dependency is given by:

$$C[qproc]' = C[qcomp1]' + (1-p[p])C[qprelim]'$$

$$= 3102.3 + (1-0.7)(397.4)$$

$$= 3102.3 + 119.2$$

$$= 3221.5 \text{ ms}$$

Chapter X

FURTHER TOPICS FOR ANALYSIS

10.1 COMPARISON OF QUERY PROCESSING TIMES

In chapter VIII, a first set of equations was developed which model the time needed to process various types of queries. In section 8.3 an example of the time to process a query was exhibited. The particular query used was one with a type A or B data dependency. Various enhancements to query processing were discussed in chapter IX. These enhancements took advantage of the modular configuration of MULTISAFE, which allow some steps to be concurrently processed. A second set of equations was developed. These equations are the enhanced versions of the equations in the first set. An example analogous to the one in section 8.3 was developed in section 9.8. This example used the enhanced version of the equation which equals the processing time for a query with a type A or B dependency.

The example from the first set of equations produced the following result:

$C[qproc] = 3533.8$ ms

The example from the enhanced set produced:

$C[qproc]' = 3221.5$ ms

In each of these examples the probability that the query passed all preliminary protection checks was 0.7. In this case, the enhancements result in an improvement of 312.3 ms in query processing time.

For a second comparison, it is arbitrarily assumed that 50% of all queries pass each of the preliminary protection checkpoints. In each of the two examples this assumption implies that the probability mentioned in the last paragraph is equal to 0.5. The values of all other variables in the two examples remain the same. With these new assumptions the following result representing the first set of equations is obtained:

$$C[qproc] = 397.4 + (0.5)(4480.6)$$
$$= 397.4 + 2240.3$$
$$= 2637.7 \text{ ms}$$

To calculate a result representing the enhanced set of equations, the value for C[qcomp1]' in section 9.8 must be recalculated using the new probability value of 0.5 in place of 0.7. Beginning from an intermediate step in the calculation of C[qcomp1]':

$$C[qcomp1]' = 0.6 + (0.5)(4431)$$
$$= 0.6 + 2215.5$$
$$= 2216.1 \text{ ms}$$

Substituting this value, and the new 0.5 probability, into the equation for C[qproc]' produces:

$$C[qproc]' = 2216.1 + (1-0.5)(397.4)$$

$$= 2216.1 + 198.7$$

$$= 2414.8 \text{ ms}$$

The MULTISAFE enhancements show an improvement of 222.9 ms in query processing time for this case.

The figures in the above two comparison cases present some numerical evidence that the enhancements developed in chapter IX do reduce the time needed to process user queries completely.

10.2 QUERY PROCESSING WITHOUT PROTECTION

MULTISAFE is designed to provide securely controlled database access. It does so by applying various enforcement policies to users who attempt to log in and who submit queries to access data. All of the equations which have been developed in previous chapters contain expressions which show the time needed to enforce data protection. To demonstrate the effect which data protection has on processing time, an equation is developed in this section which represents the time to process a typical query without performing any protection checks.

The assumptions which were enumerated in chapter VI apply to the equation developed in this section, just as they did to all previous equations. The query is assumed to

have a user-defined predicate which allows the SRM to select a B-tree index into the requested relation. Since no protection checks are being applied, there is no applicable access condition to consider.

The steps for query processing are as follows:

1. The query is parsed and then sent to the SRM.
2. The SRM retrieves tuples from the requested relation. It is assumed that the DBSD has a tuple retrieval time of R[DESD], and an index with d levels. One tuple is retrieved from the DBSD. Using this tuple, the proper relation directory, which has a tuple retrieval time of R[RD], is accessed. The tuple retrieved from the RD leads the SRM to the proper B-tree index associated with the requested data relation. It is assumed that y tuples are retrieved from the requested relation using a z-level index. The requested relation is assumed to have a tuple retrieval time of R. As each tuple is retrieved, it is immediately transmitted to the user. Tuple transmission is performed concurrently with tuple retrieval. When the last tuple is retrieved, it too is transmitted to the user. This last transmission requires one memory to memory transfer variable M to appear in the equation.

The equation which results from these steps is:

$$C[noprot] = P + (1+d)R[DBSD] + R[RD] + (y+z)R + M \qquad (10.1)$$

The reader can compare this equation with equation 9.3.

In section 9.8, an example was developed which showed that the time to process a query with a type A or B data dependency, given the selected parameters, was 3221.5 ms. Using equivalent values, make the following assignments to the variables in equation 10.1:

processing time:	P	= 0.6 ms
number of levels in the DBSD index:	d	= 3 levels
tuple retrieval time from the DBSD:	R[DBSD]	= 60 ms
tuple retrieval time from the RD:	R[RD]	= 65 ms
number of tuples retrieved from the requested relation:	y	= 50 tuples
number of levels in an index into the requested relation:	z	= 5 levels
tuple retrieval time from the requested relation:	R	= 75 ms
time for a memory to memory transfer of a tuple:	M	= 2 ms

Substituting these values into equation 10.1 produces:

$$C[noprot] = 0.6 + (1+3)60 + 65 + (50+5)75 + 2$$

$$= 0.6 + 240 + 65 + 4125 + 2$$

$$= 4432.6 \text{ ms}$$

An increase of 1211.1 ms in processing time is observed when no protection checks are applied. This may seem to be an unexpected result. The reason that the processing time for

the query with protection checks is lower becomes clear by examining the example in section 9.8 and equation 9.3. In that example, the probability was only 0.7 that query processing was carried through to completion. Otherwise, query processing time became a function of preliminary processing. Preliminary processing, represented by equation 9.2, took only 397.4 ms in the section 9.8 example. One could reduce equation 9.3 to equation 10.1 by

1. Defining p[p], the probability that a query successfully passes through the preliminary processing steps, to be 1.0.
2. Defining p[t], the probability that the last retrieved tuple is transmitted to the user, to be 1.0.
3. Combining the first two instances of the variable P into one instance of P.

With these three conditions, the times to process a query with a type A or B dependency and a query with no applied data protection are about equal. However, these three conditions are not realistic when a large sample of queries is considered. Chances are that some of these queries will be rejected during preliminary processing.

As another example of query processing taking less time with protection checks applied than without protection, con-

sider a MULTISAFE system which applies a partial enforcement policy to retrieved data. As stated in section 3.5.1, partial enforcement is a policy which permits a user to be sent all individual query responses that pass all applicable access conditions.

Suppose a user submits a query which covers y tuples in a relation. In the case where no protection of data exists, the SRM retrieves all of the y tuples and sends them to the user. In the case where protection checks are made, assume that one or more access conditions exist which limit the user to y' of the tuples, where $y' < y$. Under a partial enforcement policy, these y' tuples are the only tuples that the SRM must retrieve and send to the user. Thus, $y - y'$ fewer tuple retrievals are made with protection checks applied, and query processing in this case takes less time than when data protection does not exist.

From the above discussion, the following conclusions can be made:

1. Given the assumptions made in previous chapters and applied to all the equations which were developed, the time to process an arbitrary query with a type A or B dependency, with all protection checks in force, is less than the time to process a query with no data protection. Queries with other types of dependencies may, however, take

longer to process than the query with no data protection. Consider, for example, the query with a class C dependency (equations 9.4 and 9.5). Its equation has several terms in it that have to do with accesses to a second relation. These terms may add enough time units to that equation so that it does indeed become larger than equation 10.1.

2. The time to process a particular query with a type A or B dependency where p[p] = p[t] = 1.0 is about the same as that for the query with no protection applied.
3. The time to process some queries with protection checks in force, and with a partial enforcement policy on retrieved data in effect, is less than the time to process the same queries without data protection.

10.3 AN ALTERNATIVE ENHANCEMENT ASSUMPTION

In the process of describing the enhancements to query processing which can occur as a result of the modular configuration of MULTISAFE, an assumption was made about preliminary query processing and tuple retrieval. As discussed in section 9.1, parsed copies of a query are sent to the PSM and the SRM at the same time. The SRM begins to

retrieve tuples from the requested relation, while at the same time the PSM begins the preliminary processing of the query. The assumption which was made was that the PSM would complete its processing before the SRM completed its tuple retrieval. Furthermore, the PSM could process all the tuples already retrieved by the SRM until a point is reached where the PSM is waiting for the next tuple to be retrieved. As soon as a tuple is retrieved it is completely processed by the PSM. The PSM then waits for the next tuple to process.

In this section, instead of making the above assumption, three cases are considered. It is still assumed that the SRM and the PSM begin their respective tasks at the same time, and work concurrently on the user's query. Different combinations of SRM and PSM completion times are discussed in the cases. The effects on the enhanced equations are considered as well.

Case 1: The PSM completes its preliminary processing, and processes all tuples already retrieved before the SRM can finish its tuple retrieval. This is the situation that was explored in section 9.1. Figure 4 illustrates the relationship of the completion times of the SRM and the PSM via a time line drawing. Time moves forward from left to right in the figure. It starts at time 0 and can be assumed to

end after so many milliseconds. The reader will note that the time line for the PSM extends just beyond the one for the SRM. The extra time for the PSM is to allow for the processing of the last tuple retrieved. For some queries, the time to compute aggregate totals, render an access decision, and transmit tuples to the user might be included in the extra time as well (see, for example, section 9.4).

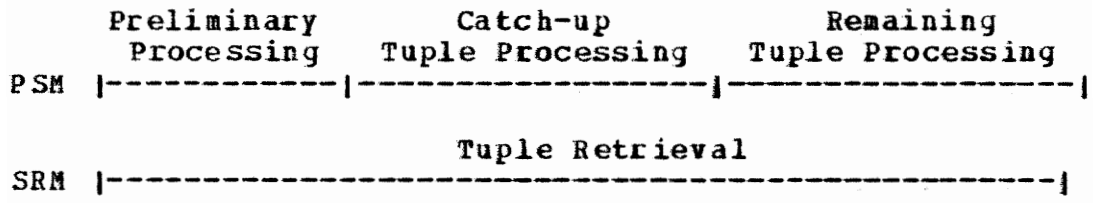


Figure 4: Time lines for case 1.

The equation for query processing is

$$C[qproc]' = C[qcompn]' + (1-p[p])\,C[qprelim]' \qquad (10.2)$$

This equation appeared throughout chapter IX, but with a digit in place of the "n" in "qcompn" to distinguish among queries. C[qcompn]' represents the time to complete query processing assuming that all steps of preliminary processing are passed. The variable p[p] is the probability that all preliminary steps are passed. Finally, C[qprelim]' is the time to perform preliminary processing.

Case 2: The PSM completes preliminary processing before the SRM finishes retrieving tuples. While the PSM is processing the tuples which have already been retrieved, the SRM completes its task. Figure 5 illustrates this case.

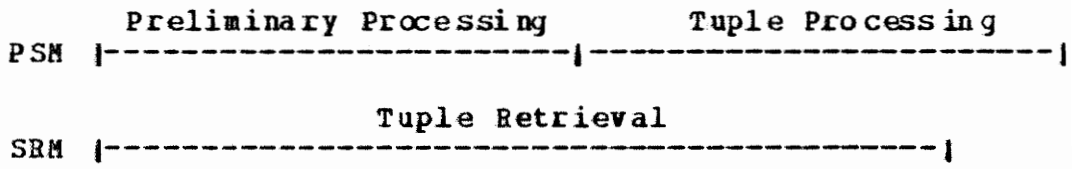


Figure 5: Time lines for case 2.

This case can be considered from the point of view of the PSM. The steps for query processing are then:

1. The PSM completes preliminary query processing.
2. If the query successfully passed all preliminary steps, then all the retrieved tuples are processed.

These steps result in the following equation:

$$C[qproc]' = C[qprelim]' + C[qcompn]' \qquad (10.3)$$

Included within C[qcompn]' is the probability p[p] that all the preliminary processing checkpoints were passed by the query.

Case 3: The SRM completes its tuple retrievals before the PSM and finishes its preliminary query processing. Figure 6 illustrates this case.

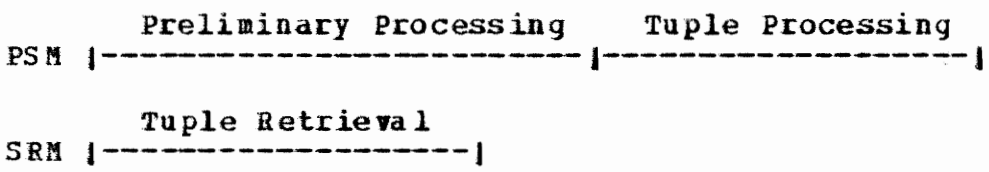


Figure 6: Time lines for case 3.

Viewed from the PSM, the steps which comprise query processing are exactly the same as those for case 2. Thus, equation 10.3 is representative of this case.

Cases 2 and 3 both used equation 10.3, so they will be combined and referred to henceforth as case 2.

Let p[orig] be the probability that the (original) situation in case 1 occurs. Then 1-p[orig] is the probability that the situation in case 2 occurs. Multiplying the probabilities with their respective equations yields the following equation:

$$
\begin{aligned}
C[qproc]'' &= p[orig](C[qcompn]' + (1-p[p])C[qprelim]') \\
&\quad + (1-p[orig])(C[qprelim]' + C[qcompn]') \\
&= (p[orig] + 1-p[orig])C[qcompn]' \\
&\quad + (p[orig](1-p[p]) + (1-p[orig]))C[qprelim]' \\
&= C[qcompn]' + (p[orig]-p[orig]p[p] \\
&\quad + 1-p[orig])C[qprelim]' \\
&= C[qcompn]' \\
&\quad + (1-p[orig]p[p])C[qprelim]' \qquad (10.4)
\end{aligned}
$$

Equation 10.4 is the time to completely process a query taking into account all possible combinations of PSM and SRM completion times during concurrent processing of the query.

Comparing equation 10.4 with equation 10.2, it becomes apparent that by assigning the probability 1 to p[orig], equation 10.4 reduces to equation 10.2. This result is

expected, since equation 10.2 is the time to completely process a query given the case 1 situation, and p[orig] is the probability that the case 1 situation occurs.

10.4 EXAMPLE

In this example, values used in section 9.8 are used for the variables in equation 10.4. Section 9.8 is an example of the enhanced equation to completely process a query with a type A or B data dependency (equation 9.3). In the section 9.8 example, the following values were either computed or taken from other examples:

time to process a type A or B query assuming that all preliminary processing steps are passed:	C[qcomp1]' = 3102.3 ms
time for the PSM to process the preliminary protection checks:	C[qprelim]' = 397.4 ms
probability that the query passes all PSM protection checks:	p[p] = 0.7

Since a type A or B query is being considered, the variable C[qcomp1]' is being used in place of C[qcompn]' in equation 10.4.

For the sake of the example, it is arbitrarily assumed that:

p[orig] = 0.5

This assumption implies that for 50% of the queries processed by MULTISAFE, the PSM finishes its protection checks and "catches up" with the SRM, as described in case 1 of the previous section.

Given the above values, equation 10.4 is given by:

$$C[qproc]'' = C[qcompn]' + (1-p[orig]p[p])C[qprelim]'$$
$$= 3102.3 + (1-(0.5)(0.7))397.4$$
$$= 3102.3 + (0.65)397.4$$
$$= 3102.3 + 258.3$$
$$= 3360.6 \text{ ms}$$

Chapter XI

SUMMARY OF CONCLUSIONS

This chapter contains a summary of the conclusions that were discussed in previous chapters. Most of these conclusions were discussed in chapter X, although some points involving several of the examples are made here as well. The last section of this chapter talks about some areas in the MULTISAFE system where future work could be pursued.

11.1 SYSTEM BOTTLENECKS

Examination of the many examples in this paper shows that one major bottleneck in MULTISAFE is the time needed to retrieve a tuple from secondary (disk) storage. The values used in the various examples were selected with the aid of [BANEJ78], [DONOJ72], [HAMAV79], and [WIEDG77]. These values thus represent possible real-world values for an operational MULTISAFE system. The slower speed of secondary storage compared to primary memory speed is felt more strongly in the equations where the SRM accesses two relations. These equations are equations 9.5 (section 9.2), 9.11 (section 9.5), and 9.15 (section 9.7). Query response time is slower in these equations as well as in any equation where a large amount of tuples must be retrieved by the SRM.

Some possible enhancements which would help to increase the speed of tuple retrieval include faster secondary storage devices, paging systems with cache memories, and disk storage schemes that place related pieces of information closer together. The latter enhancement helps to increase the chance that one disk access will retrieve several desired tuples, thus reducing the number of accesses needed to retrieve a complete set of tuples.

11.2 THE ENHANCED EQUATIONS

In chapter VIII, equations were developed which modeled query processing as a series of steps. These equations were enhanced and restructured in chapter IX to allow for concurrent execution of the PSM and SRM. In section 10.1, some numerical evidence was presented which shows that the concurrent execution of the two modules does reduce the amount of time needed to process certain queries. At the very least, the concurrency allows for increased security checking based on complex protection policies and a broad range of access decision dependencies without substantially increasing query response time.

11.3 PROTECTION VERSUS NO PROTECTION

In section 10.2 an equation was developed which models how MULTISAFE would process a query in an environment where no data protection exists. An example was produced, and it was found that the no-protection query took longer to process than a corresponding query with a type A or B data dependency with protection checks applied. This result reflected the fact that, with protection checks, some queries are rejected rather quickly by the PSM. Such queries tend to reduce the average processing time of all queries with the same type data dependency. This result also shows the advantages of a protection processor such as the PSM that can work concurrently with a tuple retrieval processor such as the SRM.

Next, a more general example was developed which showed another situation where query processing takes longer with no data protection than when data protection exists. It was found that if a partial enforcement policy exists with respect to data retrieval, then some queries caused fewer tuples to be retrieved by the SRM than if no protection policies existed.

The above discussion, and the discussion in section 11.2, may make it seem as though protection in MULTISAFE has

been implemented with little or no cost. This view is, of course, not true. Data protection always has its price. A system with data protection is more complex than a similar system without data protection. A system with protection has more software in its total package of operating system software. Execution of protection functions adds to the system workload, which can degrade the performance of other system functions and increase overall computing time in the system. Finally, there is the PSM itself. Implementation of protection in MULTISAFE is accomplished by the PSM, a processor that is not needed if there is no data protection to enforce (the reader will note that, in the steps for processing the no protection query in section 10.2, the PSM was not mentioned). The PSM could be a small, relatively inexpensive processor. During the times when it is idle, it could be used for other system functions such as backup file processing and journal trials. It has been stated many times in this paper that placing the PSM in concurrent operation with the SRM has much less impact on query processing than having the PSM and SRM connected in series. Nevertheless, the PSM is another piece of hardware to be added to MULTISAFE which increases the total cost of the MULTISAFE system in terms of hardware and the operating system software needed to make the hardware work.

11.4 PSM AND SRM COMPLETION TIMES

In section 10.3 an alternate assumption was made to one that was discussed in section 9.1. In section 9.1, it was assumed that the PSM would finish its protection checks before the SRM would finish its tuple retrieval during processing of a query. Furthermore, the PSM could completely process all of the tuples that the SRM has retrieved so far, again before the SRM would finish its task. Section 10.3 did not make the assumption of section 9.1, but instead analyzed all possible PSM and SRM completion times. Equation 10.4 was developed as a result of this analysis.

Equation 10.4 probably models the real-world situation in a MULTISAFE system more closely than its counterpart, equation 10.2. In fact, there are several assumptions that were made throughout this paper that may have narrowed the scope of situations that the equations in this paper properly model. However, it was never the purpose of this paper to model all possible real-world situations. Several reasonable assumptions were made so that the analyses in this paper could be performed, and so that evidence could be gathered to support one of the main hypotheses in the paper; that a dedicated protection processor working concurrently with a tuple retrieval processor would have a minimum amount of impact on a data protection system.

11.5 FUTURE WORK

Items of interest that would make useful future follow-on studies to the work in this paper include the following:

1. A simulation study. Such a study is being worked on now [TALBT81] by simulating the processors in MULTISAFE through software. It might be useful to extend this study by constructing a MULTISAFE system with hardward processors, and then writing the necessary software to simulate query processing.
2. A study involving a performance equation analysis of a multi-user MULTISAFE system. Such a study would have to include a queueing analysis to model the competition for the resources in MULTISAFE among several users.

BIBLIOGRAPHY

ASTRM76 Astrahan, M. M., et. al., "System R: Relational approach to Database Management", ACM Transactions on Database Systems, 1, 2, June, 1976, pp. 67-137.

BALLE81 Balliet, Earl J., "Modeling of MULTISAFE Protection Enforcement Processes with Extended Petri Nets", M. S. Thesis, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Va., 24061 (January 1981).

BANEJ78 Banerjee, Jayanta, and Hsiao, David K., "Performance Study of a Database Machine in Supporting Relational Databases", Proceedings of Fourth International Conference on Very Large Data Bases, ed. by S. Bing Yao, West-Berlin, Germany, Sept., 1978, pp. 319-329.

CHAMD76 Chamberlin, D. D., et. al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control", IBM Journal of Research and Development, 20, 6 (November 1976), pp. 560-575.

COMED76 Comer, Douglas, "The Ubiquitous B-Tree", ACM Computing Surveys, 11, 2, June, 1976, pp. 121-137.

CONWR72a Conway, Richard W., W. L. Maxwell and Howard L. Morgan, "On the Implementation of Security Measures in Information Systems", Communications of the ACM, 15, 4 (April 1972), pp. 211-220.

CONWR72b Conway, Richard, Maxwell, William, and Morgan, Howard, "Selective Capabilities in ASAP--A File Management System", Proc. of the SJCC (1972), pp. 1181-1185.

DENND79 Denning, Dorothy E., Peter J. Denning, and Mayer D. Schwartz, "The Tracker: A Threat to Statistical Database Security", ACM Trans. on Database Systems 4, 1 (March 1979), pp. 76-96.

DOBKD79 Dobkin, David, Anita K. Jones, and Richard J. Lipton, "Secure Databases: Protection Against User Inference", ACM Trans. on Database Systems 4, 1 (March 1979), pp. 97-106.

DONOJ72 Donovan, John J., Systems Programming, New York, McGraw-Hill Book Company, 1972, p. 96.

GRIFP76 Griffiths, Patricia P. and Wade, Bradford W., "An Authorization Mechanism for a Relational Database System", ACM Trans. on Database Systems, 1, 3 (September 1976), pp. 242-255.

HAMAV79 Hamacher, V. Carl, Vranesic, Zvonko G., and Zaky, Safwat G., Computer Organization, New York, McGraw-Hill Book Company, 1979, pp. 289-290.

HARTH75 Hartson, H. Rex, "Languages for Specifying Protection Requirements in Data Base Systems--A Semantic Model", Ph.D. Dissertation, Dept. of Computer and Information Science, The Ohio State University (August 1975), Research report: OSU-CISRC-TR-75-6.

HARTH76a Hartson, H. Rex, and Hsiao, David K., "A Semantic Model for Data Base Protection Languages", Proceedings of the Second International Conference on Very Large Databases, North Holland Publishing Co., Brussels, Belgium, Sept., 1976.

HARTH76b Hartson, H. Rex, and Hsiao, David K., "Full Protection Specifications in the Semantic Model for Database Protection Languages", Proceedings of the ACM Annual Conference, Oct., 1976, pp. 90-95.

HARTH77 Hartson, H. Rex, "Dynamics of Database Protection Enforcement--A Preliminary Study", Proc. of the IEEE Computer and Software Applications Conf., Chicago (November 1977), pp. 349-356.

HARTH80 Hartson, H. Rex, "Implementation of Predicate-Based Protection in MULTISAFE", Technical Report CS80010-R, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Va., 24061.

HARTH81 Hartson, H. Rex, and Earl J. Balliet, "Modeling of MULTISAFE Protection Enforcement Processes with Extended Petri Nets", Technical Report CS81005-R, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Va., 24061 (March 1981).

LAMPB71 Lampson, Butler W., "Protection", Proc. Fifth Princeton Symp. on Information Sciences and Systems, Princeton University (March 1971), pp. 437-443; reprinted in ACM SIGOPS Operating Systems Review 8, 1 (January 1974), pp. 18-24.

STONM74 Stonebraker, Michael, and Eugene Wong, "Access Control in a Relational Data Base Management System by Query Modification", Proceedings of the ACM Annual Conference (November 1974), pp. 180-186.

TALBT81 Talbott, Thomas, "Implementation of MULTISAFE in a Relational Database Environment", M. S. Project Report, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Va., 24061.

TRUER80 Trueblood, Robert P., Hartson, H. Rex, and Martin, Johannes J., "MULTISAFE -- A Modular Multiprocessing Approach to Secure Database Management", Technical Report CS80008R, Dept. of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Va., Oct., 1980.

WIEDG77 Wiederhold, Gio, Database Design, New York, McGraw-Hill Book Company, 1977, pp. 28, 37-47, 56-57.

WONGE76 Wong, Eugene, and Karel Youssefi, "Decomposition--A Strategy for Query Processing", ACM Trans. on Database Systems, 1, 3 (September 1976), pp. 223-241.

APPENDIX

Variables used in performance equations.

btt -- the time to transfer a block of data from disk to primary memory.

C -- a variable set equal to various equations. The subscripts below distinguish the different occurrences of C.

auths -- the time to retrieve access condition authorization tuples from the AUTHS relation for a user's access profile.

EAC -- the time to retrieve tuples from the CONDITIONS relation in order to form the Effective Access Condition of a query.

flogin1 -- the time to process a login request that fails because a user entered incorrect login parameters.

flogin2 -- the time to process a login request that fails because no access condition authorizations exist for the logging in user.

noauths -- the time to determine that no access condition authorizations exist for a logging in user.

noprot -- the time to completely process a query when no protection checks are applied.

profile -- the time to retrieve user profile information from the USERS relation.

qcompn -- the time to complete query processing after having finished preliminary processing. The n is a number which distinguishes different query completion equations.

qproc -- the time to completely process a query.

slogin -- the time to process a successful login.

tlogin -- the time to totally process a login request.

usergps -- the time to retrieve user group information from the USERS relation.

C' -- a variable set equal to equations which model query processing taking into account various enhancements. A few of the same subscripts used with C are used with C'. One more subscript, however, is used:

qprelim -- the time to complete preliminary query processing when a query is rejected for some reason.

d -- the number of tuples retrieved from a relation.

g -- the number of tuples retrieved from a relation.

j -- the number of tuples retrieved from a relation.

k -- the number of tuples retrieved from a relation.

M -- the time to transfer a tuple from the memory in the SRM to the memory in the UAM. M includes any domain projections performed by the SRM.

m -- the number of tuples retrieved from a relation.

P -- a quantity of CPU processing time. Different types of processing which take different amounts of time are not represented by various values of P using subscripts. Rather, the different processing types are grouped together and their times represented by the variable P in many of the equations. Thus P may be the sum of the times needed to perform different types of processing, or perhaps the maximum value of these times.

p -- the probability of an event. Subscripts are used in some cases to distinguish between various probabilities.

R -- the time to retrieve one tuple from disk, given that the address of the disk block containing the tuple can be computed. Subscripts are used in some cases to identify the relation from which the tuple is retrieved.

R' -- the time to retrieve a block of data from disk given that the block's starting address is known.

R1 -- the same as R, except that no distinguishing subscripts are used.

r -- the rotational latency time; that is, the time for the beginning of a block of data to rotate under the disk read/write head.

s -- the seek time for the disk read/write head to move to the track where a block of data is located.

x -- the number of levels in a B-tree index into a relation.

x1 -- the number of levels in a B-tree index into a relation.

x2 -- the number of levels in a B-tree index into a relation.

x3 -- the number of levels in a B-tree index into a relation.

y -- the number of tuples retrieved from a relation.

y' -- the number of tuples which were retrieved from a relation that are returned to the user.

z -- the number of levels in a B-tree index.

z1 -- the number of levels in a B-tree index.

Vita

Mason C. Deaver, Jr.

360-B Glendare Drive

Winston-Salem, NC 27104

Education

High School: William Fleming Senior High School
Roanoke, Va.

Grade Point Average: 4.00/4.00 (Valedictorian)

Graduated: June, 1974

Undergraduate: Randolph-Macon College
Ashland, Va.

Grade Point Average: 3.84/4.00

Graduated: June, 1978

Degree: B.S. - Mathematics and Computer Science

Honors: Phi Beta Kappa honorary scholarship fraternity
Omicron Delta Kappa honorary leadership fraternity
Chi Beta Phi honorary scientific fraternity

Graduate: Virginia Polytechnic Institute and State University

Blacksburg, Va.

Grade Point Average: (currently) 3.55/4.00

Graduated: June, 1983

Degree: M.S. - Computer Science and Applications

Honors: Graduate Assistantship

Upsilon Pi Epsilon Honor Society in the Computing Sciences

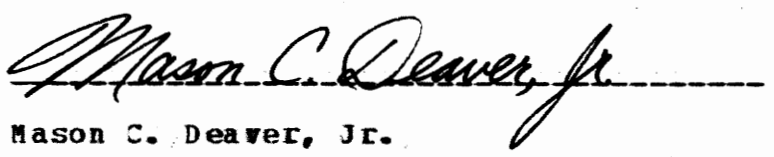
Current Employment

Western Electric

Winston-Salem, NC

Title: Information Systems Staff Member

(Programmer/Analyst)



Mason C. Deaver, Jr.