Submitted to ACM Transactions on Database Systems for publication.

Technical Report CS80008R

MULTISAFE -- A Modular Multiprocessing Approach to Secure Database Management

by

Robert P. Trueblood
H. Rex Hartson
Johannes J. Martin

October 1980

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

MULTISAFE--A MODULAR MULTIPROCESSING APPROACH TO SECURE DATABASE MANAGEMENT

by

Robert P. Trueblood*

H. Rex Hartson**

Johannes J. Martin

Virginia Polytechnic Institute and State University

Blacksburg, VA 24061

Abstract: This paper describes the configuration and inter-module communication of a MULTIprocessor system for supporting Secure Authorization with Full Enforcement (MULTISAFE) for database management. A modular architecture is described which provides secure, controlled access to shared data in a multiuser environment--with low performance penalties, even for complex protection policies. The primary mechanisms are structured and verifiable. The entire approach is immediately extendible to distributed protection of distributed data. The system includes a user and applications module (UAM), a data storage and retrieval module (SRM), and a protection and security module (PSM). The control of intermodule communication is based on a data abstraction approach. It is initially described in terms of function invocations. An implementation within a formal message system is then described. The discussion of function invocations begins with the single terminal case and extends to the multiterminal case. Some physical implementation aspects are also discussed, and some examples of message sequences are given.

Key Words and Phrases: security, protection, multiprocessing, modularity, database management, abstract data types, computer architecture, data security, inter-process communication, distributed processing

---

* Present address: Dept. of Mathematics, Computer Science, and Statistics, University of South Carolina, Columbia, SC 29208.

** This material is based upon work supported, in part, by the National Science Foundation under Grant Number MCS-7903936.

1. INTRODUCTION AND BACKGROUND

1.1 Purpose of MULTISAFE

MULTISAFE [TRUER79] is designed to provide securely controlled database access, and to do it in a way that:

1) is verifiably secure
2) does not incur a prohibitive performance penalty
3) produces a modular system in accordance with the modern structured approach to design
4) is naturally extendible to the protection of distributed data
5) provides flexible mechanisms able to adhere to complex protection policies

The MULTISAFE architecture derives performance advantages from its concurrency, and its protection processor allows for generalized protection mechanisms (such as those in [HARTH76a, HARTH76b]). Work is currently being done on the application of MULTISAFE to a distributed data environment and on the performance behavior of MULTISAFE. However, the purpose of this paper is to introduce MULTISAFE and to focus on its security related aspects.

1.2 Related Work

Functional modularization of database systems has appeared in several forms since Canaday et al. [CANAR74] introduced the "back-end" computer to do DBMS processing concurrently with a non-DBMS host processor. Bisbey and Popek [BISBR74] encapsulated the operating system and security processes on a minicomputer, separate from user and application procedures. Downs and Popek [DOWND77] placed data security functions in a "DBMS kernel," which controls all physical I/O. Some work at I. P. Sharp [GROHM76, KIRKG77a, KIRKG77b] has attempted to apply the security kernel and reference monitor concepts of operating systems directly to the database problem. Under the direction of David Hsiao, the Data Base Computer (DBC) [BANEJ78] has become an integrated system architecture for secure data management. The DBC uses specialized hardware based on near-term future technology. It combines and extends the concepts of back-end and associative processors. Lang, Fernandez, and Summers [LANGT76] have proposed a division of application software into three classes of object modules: (1) the application, (2) data interaction, and (3) data control. Cook [COOKT75] has investigated the use of separate virtual machines in a DBMS design. He proposed the concept of a "user machine" which gives each user his/her own structural view of the database.

The problem of security verification has been studied with respect to proving program correctness, primarily within operating systems.

Much of this work has concentrated on security kernels [MILLJ76, NEUMP77, ROBIL77, POPEG78, POPEG79, MCCAE79, GOLDB79, WALKB80], the objective of which is to construct a virtual machine in which security mechanisms are isolated from other mechanisms.

It is evident from the above that the concepts of isolation and separation have been considered important for supporting data security. However, the protection question in these approaches can still be considered to be open, because physical isolation is not a guarantee of security. As pointed out in a recent workshop on distributed computing (reported by Peterson in [PETEJ79]), physical isolation can provide "more apparent protection of information by providing physical control over that part of the system; information can flow in and out only over easily identified wires. This produces 'warm feelings' in the user of the system."

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 gained by isolation.

2. OVERVIEW OF THE SYSTEM ARCHITECTURE

A data management system can be functionally divided into three major modules (a module being a combination of hardware and software):

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

Each one of these modules is treated as a separate and isolated process which is connected in a precise manner to the others to form the multiprocessing system called MULTISAFE. In MULTISAFE all three modules function in a concurrent fashion. That is, the UAM coordinates and analyzes user requests at the same time that the SRM generates responses for previous requests. Simultaneously, the PSM continuously performs security checks on all activities. Figure 1 illustrates the logical relationships among the three modules.

The modules of MULTISAFE are logically separated. The correspondence from its logical separation to a physical separation (i.e., the implementation of the modules on physically distinct processors) is not critical to security, but may help improve performance by introducing actual concurrency. Further discussion of implementation will be deferred until a later section.

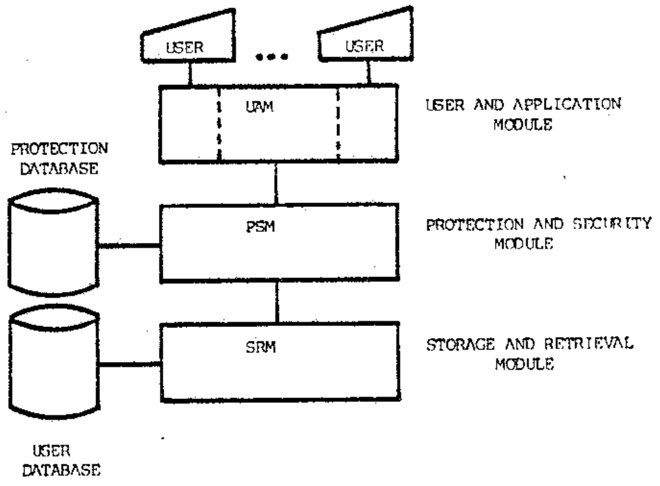

USER
USER
UAM
USER AND APPLICATION MODULE
PROTECTION DATABASE
PSM
PROTECTION AND SECURITY MODULE
SRM
STORAGE AND RETRIEVAL MODULE
USER DATABASE

Figure 1. Logical relationships among MULTISAFE modules

2.1 The Protection and Security Module (PSM)

The PSM has much in common with some of the concepts reviewed in section 1.2, in the sense that security mechanisms are encapsulated or isolated from other modules. On the other hand, the PSM also differs from most of those views, because the PSM is dedicated to security checking and is not mixed in with other operating system functions (e.g., I/O handling) or database functions. The PSM offers fine resolution (granularity) and, as a separate processor, can be sufficiently sophisticated to make data security checks in a way which can more precisely adhere to the complex protection policies needed for database systems. The PSM performs only security checks and does not perform database I/O.

The PSM can make access decisions based on three classes of dependency:

1) data-independent
2) data-definition-dependent
3) data-value-dependent

Data independent access conditions can depend on user and/or terminal identification information and dynamic system variables such as time of day and various kinds of system status information. Data-definition-dependent conditions involve attributes (attribute names), but not their values. For example, a user may have permission to access names and addresses (from an employee information file) but never salary information. Data-value-dependent conditions require the values of attributes to be checked. For example, no user may have permission to see the salary of anyone who has a value of 'manager' for the 'job title' attribute.

In addition, the PSM can perform other functions. Some of these are history keeping, auditing, integrity checking, cryptographic processing, and backup/recovery control.

2.2 The User and Applications Module (UAM)

The UAM acts as an interface between the user and the system by reading and analyzing input queries and by formatting and displaying (or printing) results. It also provides working storage and computation for the user. All security and I/O routines are removed from the UAM and are completely isolated from the user. One exception is that the UAM will provide low level primary memory management and protection for memory shared by multiple users within a single UAM. (This memory protection is accomplished by existing operating systems techniques and is not part of this present work.) Much of what is traditionally part of the operating system will execute within the UAM.

The UAM differs from its counterpart in other proposed systems in the operations that are performed. For example, in the back-end system of Canaday et al. the host performs security operations, whereas the UAM does not. In Bisbey and Popek's encapsulation approach the central computer is allowed to do I/O operations, whereas the UAM is not.

There are several ways to view the UAM in a multiuser environment. First, the UAM can be viewed as a large conventional multiprogrammed processor, with several disjoint user address spaces. An alternative view is a collection of very "intelligent" terminals each with its own private memory and processor. In this view some or all of the UAM resides in each of the terminals. With an intelligent terminal a user's software and local data buffers become physically isolated from those of other users. Further details of the UAM are not within the scope of this paper.

2.3 The Storage and Retrieval Module (SRM)

The primary task of the SRM is to perform database accesses for the UAM and PSM.

The SRM processor can be conventional computer hardware (mini- or maxi-computer) and/or conventional DBMS software. It can also be in a back-end processor or a database machine with specialized hardware for storage and retrieval operations.

Since the SRM resides on its own processor(s), it is also possible for the SRM to perform certain data manipulation operations in addition to data retrieval. That is, the SRM can compute SUM, COUNT, and AVERAGE or other special functions, such as JOIN and PROJECTION and materialization of views for a relational database.

In addition to managing database storage and retrieval operations, the SRM maintains private application files for non-DBMS users. That is, for a non-DBMS application program being processed by the UAM, the SRM performs the "simple" (e.g., reading next record on a tape) I/O operations on the file, forcing all I/O to be controlled by a common set of mechanisms.

3. THE SERVICES PROVIDED BY MULTISAFE

MULTISAFE provides secure, controlled access to shared data in a multiuser database environment. MULTISAFE serves two classes of users: authorizers and data accessers. Correspondingly, the PSM has two parts: the authorization process and the enforcement process. In reality, the authorizers are probably a subset of the data accessers. However, they can be treated as two distinct sets, since at any given time the roles will always be distinguishable. For both groups MULTISAFE provides secure login and logout, including user identification and authentication. Initially, only one authorizer (the system administrator: SYSADMIN) exists. The SYSADMIN can designate users (create accounts) and, by issuing authorizations, can grant them access rights to various parts of the database. The SYSADMIN can also designate other authorizers.

The authorization process provides the ability to add, delete, modify, and display authorization information stored in the Protection Database (see Figure 1). The right to make, modify, or see authorizations depends on the concept of "ownership." An authorizer becomes an

4. ASSUMPTIONS AND DEFINITION OF SECURITY

Several assumptions which help to focus this work are as follows:

Assumption 1. Controlled Physical Access
The system is accessed only via terminals. Consider that the system is physically protected by an impenetrable wall with small holes through which wires protrude to terminals in the outside world. Any information or signals can be sent in through those wires. If the modules of MULTISAFE are physically distributed, the equivalent of the impenetrable wall can be provided by anti-eavesdropping techniques such as encryption.

Assumption 2. PSM Programming Impervious to Modification
PSM programming is built into a Programmable Read-Only Memory (PROM). It is physically impossible for PSM programs to be modified by a user, via a terminal.

Assumption 3. Correct User Identification
User identification (authentication) will be assumed to be done correctly. User identification is being attacked elsewhere as a completely separate problem [EVANA74, PURDG74, COTTI77]. Further, it is assumed that user identification can be reauthenticated whenever necessary or desirable, so that the relationship between user and terminal remains constant to MULTISAFE.

Assumption 4. Separation of Users in UAM
The UAM provides ordinary primary memory protection, so that multiple users are prevented from interfering with each other's processes, data, or messages.

Assumption 5. Limitation of Scope
Security in this work refers to access controls, and not information flow controls [DENND76] or inference controls [HOFFL70, SCHLJ75, SCHWM79, DOBKD79, DENND79, KAMJB77]. The flexibility of a generalized PSM processor, of course, admits to future addition of these and other controls.

Assumption 6. Discretionary Access Control
Discretionary authorization is assumed, being a more general case than non-discretionary (security levels), but not ruling out non-discretionary policies. An important implication is that many users are typically also authorizers.

All that follows, particularly the definition of security and its constituent conditions, will apply to systems subject to the constraints of these assumptions. The definition of data security in this work emphasizes security explicitly as a relationship between authorization and enforcement:

> Definition 1: A system is data secure if, in that system, the enforcement process allows the system to perform only those access operations which are specified by the authorizers.

At this point, it is useful to have a clear understanding of the term "access." In this work the following definition of access is used.

> Definition 2: Access includes all operations used for reading, writing, or modifying data stored in the system.

Definition 1 can be restated as a set of four conditions:

Condition 1. Correctness of Authorization Process
All authorizations specified by the authorizers (and only by proper authorizers) are properly stored in the PSM Protection Database.

Condition 2. Correctness of Enforcement Process
All access decisions made by the PSM are correct with respect to (1) the access request, (2) the stored authorization information, and (3) the system state, including data values, at the time of the decision.

Condition 3. Complete Mediation
All access requests are subject to enforcement (access decision by the PSM).

Condition 4. Prohibition Against Spurious Data Transmission
No data may move between a user and the database (in either direction), except as a correct response to an access request.

These conditions are intuitively shown to completely embody Definition 1 as follows. (The informality of the definition and the conditions precludes a formal proof of completeness.) Condition 4 states that every data access is in response to an access request. (The requirement that it be a correct response also eliminates secondary trickery, such as a "Trojan Horse" in the SRM trying to deceive the PSM by sending prohibited data disguised as the response to some other request.) By condition 3, then, every data access that occurs is subject to an enforcement decision. Condition 2 implies that every data access is subject to an access decision that is correct with respect to the stored authorization information. Finally, by condition 1, every data access is subject to an access decision that is correct with respect to a proper authorizer's specifications of access privileges, and this is a restatement of Definition 1.

In this paper correctness of the authorization process and the enforcement process (conditions 1 and 2) will be assumed. This could be shown, given a formal model of authorization and enforcement such as is found in [HARTH76a]. Conditions 3 and 4 are of interest here, as these deal with intermodule communication--message content and message paths--which controls the sequences by which modules invoke each other's functions.

5. INVOCATION STRUCTURE AND MESSAGE SECURITY

The problem of message security is now approached in two stages. First, discussion is focused on a simple system that supports only one terminal; then the solution obtained is adapted to the general system of arbitrarily many terminals. In this section the problem of intermodule communication is abstracted to a discussion of function invocations, to eliminate implementation details involved in a message system that might be used to deliver the calls and returns from module to module. A description of such a message system follows in later sections.

5.1 The Single Terminal Case

The system discussed first is simplified in that it does not support simultaneous access of several users. Of course, many users can access the system and share data, but only through the same terminal, that is, not simultaneously but one after another. Discussing this simplified system first permits treatment of the enforcement mechanism for access rights independently from the problem of separating simultaneous processes.

Figure 1 reveals that the system is composed of a sensitive part (the SRM and its protected data) and a nonsensitive part (the UAM and the users), and the focus here is concentrated on the communication of the nonsensitive part with the sensitive one. The PSM can be interpreted as the gate to and from the sensitive part; all messages between the two parts have to pass through the PSM.

The basic pattern of information flow between the UAM/user portion of the system and the SRM is reflected in the following program executed by the PSM. The program is invoked upon a LOGIN request from the UAM and remains to be the monitor of all transactions until a LOGOUT request is received and executed.

```
PSM_P:   PROC;
ID := USERID; /* USERID is a function that converses with
                 the user in order to establish her identity.
                 ID is the internal code for the user and
                 the basis for all later access decisions. */
IF VALID(ID)
  THEN BEGIN
          Q := USER_REQUEST /* prompts and obtains request
                                   from user via the UAM */;
          WHILE NOT LOGOUT(Q)
             DO IF CHECK1(Q, ID) /* may ID ask for Q?
                                     (data independnent checking) */
                  THEN BEGIN
                          R := SERVICE(Q) /* obtain response
                                             to request Q from SRM.
```

```
                                      Potentially, R is a file
                                      that consists of many
                                      blocks */;
                              RU := EXTRACT(R,ID) /* extract what
                                      ID may see of R (data
                                      dependent checking and
                                      partial enforcement) */
                              IF IS_EMPTY(RU)
                                 THEN CALL DENIAL(Q)
                                 ELSE CALL RESPONSE(RU);
                             END;
                        ELSE CALL DENIAL(Q);
                     Q := USER_REQUEST;
                   END;
              CALL LOGOUT_RESPONSE;
         ELSE CALL LOGIN_DENIAL;
       RETURN;
     END;
```

The function USERID determines, possibly by a probing conversation, the user's identity. It may also inform the UAM whether the user has logged on successfully and, thus, established system occupancy. The variable ID, local to the PSM, is then used for all later security checks.

The function USER_REQUEST prompts the user and obtains his next service request. If this is not a LOGOUT request then it will be checked (by CHECK1) against the user id to determine whether it is authorized.

Although some of the subprograms used in PSM_P converse either with the user (USER_REQUEST, RESPONSE, DENIAL) or with the SRM (SERVICE), none converses with both; the only information path between the UAM and the SRM is established by the sequence

```
Q := USER_REQUEST;
CHECK1(Q, ID);
R := SERVICE(Q);
RU := EXTRACT(R,ID);
CALL RESPONSE(RU);
```

where a security check is performed on both the information Q that enters the sensitive area and the information R that leaves it.

The security of the system is therefore determined by the properties of CHECK1 and EXTRACT. The correctness of these is assumed here (conditions 1 and 2 of section 4).

5.2 The Multi-Terminal Case

If more than one terminal is attached to the system, there is the problem of keeping interactions originated by different terminals separate. This is partially accomplished by tagging all messages that belong to the interaction originated by a particular terminal t[i] with the identification of t[i], say, i. It may be assumed, for example, that the UAM contains a polling loop that interrogates, in turn, all terminals, picks up existing messages and tags them with the terminal identification code. These terminal id's accompany all communication messages exchanged among the different modules and finally, for output messages, determine at which terminal the message is to be displayed.

In order to ensure that no confusion arises among the different processes, simultaneous and pseudo-simultaneous, a mechanism is needed that uses the terminal id's for controlling the association of messages and processes.

This mechanism will now be described. It has two important properties:

a) It is transparent to the programs written for the different modules. This, for example, makes it possible to use the program PSM_P, described above, for a multiple terminal system without alteration. The advantage of maintaining its simplicity and, hence, clarity and verifiablity is obvious.

b) The mechanism is the same for all modules. Thus, verified once, its correctness is guaranteed wherever it is used.

The mechanism consists of a calling sequence (invoked by all function or subroutine references to programs in other modules), a return sequence (invoked by the return statement of a program that was called from another module) and a control loop. Each module owns a process table and an input queue, devices needed by the communication mechanism but invisible to the other module programs.

The input queue may accomodate the requirements of a priority scheduler and need not follow the strict queuing discipline. It must store tuples used by the calling and returning sequences described below, of either two or four components. Any module of the system may feed a queue but only the module that owns it may inspect or remove members of the queue.

The process table permits a module to file the environment of a program (local variables, return address, etc.) under the terminal id. The mechanism is now described informally, starting with the calling sequence.

The Calling Sequence

When a program F1 that executes in module i needs to call a program F2 in module j with the parameter set P—invoked formally by F2j(P)—the following events occur (see Figure 2):

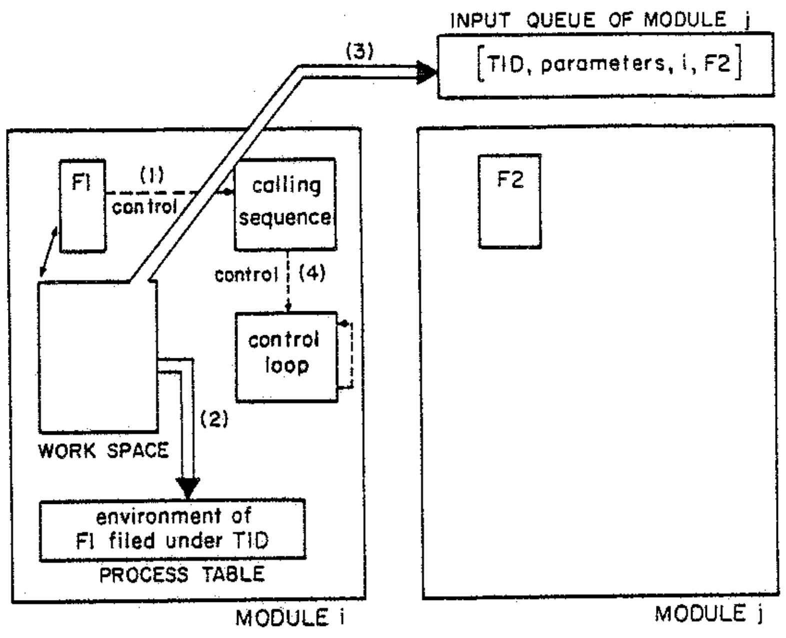


Figure 2. The calling sequence

(1) The calling sequence is activated, which then

(2) files the environment of F1 (including its return address) under the current terminal id, in the process table of module i,

(3) puts the 4-tuple [terminal id, P, i, F2] onto the input queue of module j (see next section for explanation), and

(4) transfers control to the contol loop (see next section) of module i.

The Control Loop

The control loop of module i (active only if no other program is being executed in module i) removes the 'next' item from its input queue (it waits if there is none). It then determines whether the item is a reference from another module to a function in module i or whether it is the response to a function reference previously issued by module i itself, now being returned from another module. If the item is a function reference, then it is a 4-tuple (see description of the 'calling sequence' above) of the form:

[terminal id, parameter set, source module id, function name].

In this case, the control loop (see Figure 3a) sets up a skeleton environment (1) for the program called and inserts the terminal id and the source module code into it. It then transfers control (2) to the function requested with the parameter set given. The control loop itself relinquishes control.

If the item found on the queue is the response to (that is the return from) a function reference, then the item is the pair (see the next paragraph on the return sequence):

[terminal id, result set]

In this case, the control loop (see Figure 3b) reinstates the environment retrieved from the process table under the terminal id (1), depo-

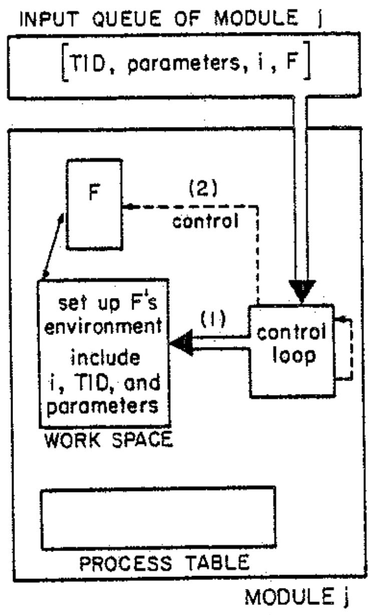


Figure 3a. The control loop (calling)

sits the result set (2) at the place ordinarily used by functions for returning their results and transfers control (3) to the return address (also retrieved from the process table).

The Return Sequence

When a function that executes in, say, module j issues a RETURN(result) statement the following happens (see Figure 4):

(1) The return sequence is activated, which

(2) enters the pair: [terminal id, result set]

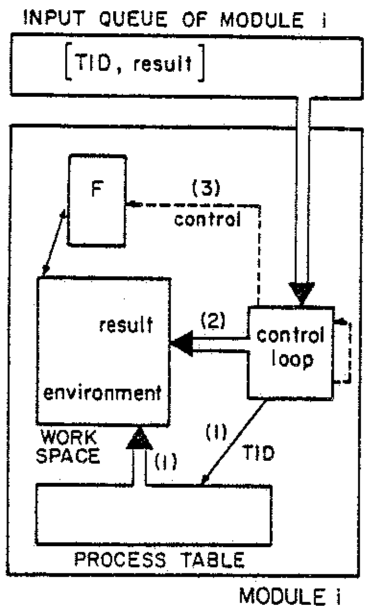


Figure 3b. The control loop (returning)

into the queue of the source module i, (recall that the terminal id and the source module code are part of the function's environment) and

(3) transfers control to the control loop of module j.

The coding of these three operations is straightforward and the verification of their correctness can easily be achieved by axiomatizing the queue and process table operations using the technique of algebraic specifation [GUTTJ78].

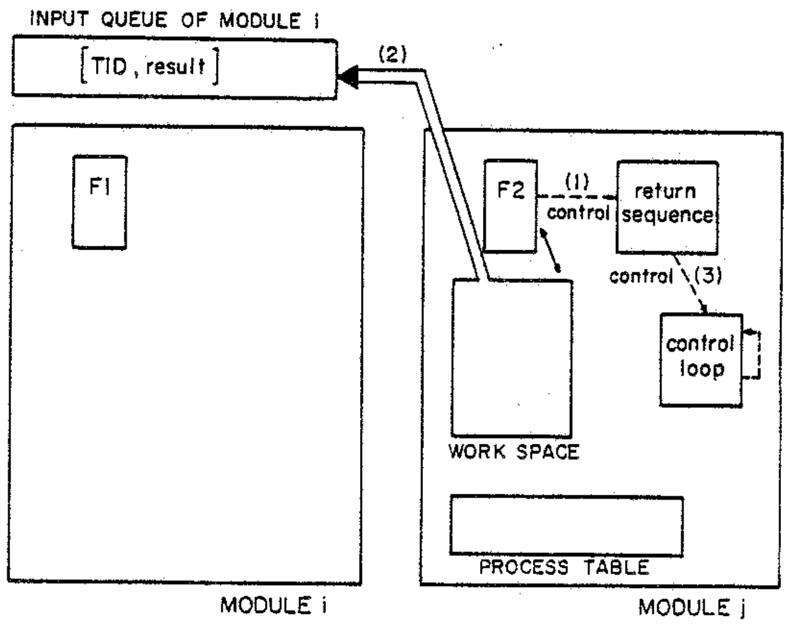


Figure 4. The return sequence

Simultaneous Transactions by the Same User

The mechanism described above prevents a transaction from influencing any other transaction, provided that the terminal ids are unique. This uniqueness, however, may not always be ensured for the following reasons:

1) The user might enter a new request before the previous one has been fully processed. Thus, if the terminal id consists of only the terminal number, two or more message sequences could build up which would interact causing undesirable results. There are two basic solutions to this problem:

a) The situation would be detected by discovering that the process table of the module addressed already contains an entry that indicates a pending process for the given terminal id. (Recall that a control loop, which picks up all messages, can only be active if all processes of the corresponding module are dormant, that is, have their environments stored in the process table.) Immediately, an alternate function could send a message back to the user asking for patience. This process would not alter the current state of the process table and, thus, leave the ongoing process undisturbed.

b) The second message could be forced to have a different terminal id by qualifying the terminal id with the addition of a time stamp. Now, the two requests would be treated independently, in fact, since the communication mechanism is transparent to the program that invokes it, such a program has no means of discovering that more than one message is currently processed. This method is not recommended for the following reason. Letting the user issue new requests into a pending process is meaningful only if a related sequence of transactions (such as a question and answer interchange) is to be established between the module and the user. The apparently different tasks would create different environments preventing the desired coherent discourse. Such a discourse is, for example, needed for the LOGIN process and it may also be desired for request cancellations and modificatons.

The problem could be overcome by giving up some of the transparency of the communication process and allowing the higher level programs to examine the terminal id. However, this generally is undesirable because it undermines the simplicity of the overall scheme in an essential way.

2) The second situation where interference could occur arises if a more complex processing pattern is needed. Suppose a module A calls a function executed in module B which, in turn, calls a function performed by module A. (This could happen rather often.) Since all processes have the same terminal id the last process destroys the environment of the previous one. This problem is overcome, however, by attaching a level number to the terminal id. The calling sequence increments the level number before putting the information packet on the queue of the next module and the return sequence decrements it.

6. INTERMODULE COMMUNICATION

6.1 Introduction

The purpose of this paper is to describe logical concepts, not to provide a blueprint for a physical implementation. Whereas more specific physical views are required to discuss performance and cost, a logical view is often most suitable for discussing security. An actual implementation will translate the logical view into one of several possible physical manifestations. It is crucial to guarantee that, during this translation, the logical characteristics are preserved with regard to security.

For many readers, however, it is helpful to supplement an abstract discussion with a concrete example in order to convey a more complete understanding of the kind of system being described. This section provides an example of an implementation with physically separate (but not distributed geographically) processors.

6.2 Procedure Calls as Messages

It has been suggested [LAUEH78, as described by Sturgis in PETEJ79] that message passing and procedure calling are essentially equivalent constructs for communication within a single processor system. The discussion in [PETEJ79] further suggests that this duality between calls and messages may not apply as well to distributed systems. The apparent difficulty arises in a pipeline situation (for example) where information flow continues in one direction. However, in any implementation, a response message or return to a call is eventually necessary, if only to acknowledge the request message or call (i.e., a pipeline eventually flows back to return the resulting data). If the messages or calls do not require an immediate reply, they can be stacked or nested and replies can be delayed until the requested data can be sent back to the requester. In such a case, the duality appears to stand. In MULTISAFE procedure calls and returns are conveyed by messages (for requests and responses), but immediate responses are not required for request messages. Further, there can be concurrency between those messages which are simple acknowledgements and those which contain data (discussed in sections and). Typed languages with special facilities for sending and receiving messages [HUNTJ79] are interesting to consider for the construction of a system such as MULTISAFE.

There is, of course, a correspondence between the functional request and response roles of the messages (described later in section) and the call and return mechanisms of the procedures (described in section 5). However, because the messages and the function calls are different kinds of abstractions, the correspondence is not one-to-one. In particular, it is useful for the PSM to remain "in charge;" i.e., to "solicit" requests from the UAM with a USER_REQUEST call. Then, no unsolicited requests can be autonomously generated by the UAM. (It is generally the case in most systems, anyway, that the system signifies its readiness to receive requests.) The user states his/her request (via the UAM) in the return to that call. The relationship between a request and its response is discussed further in the section on message sequences.

6.3 Message Structure

Messages are used to carry function calls (requests) and returns (responses) between modules. Recall the form of the function reference 4-tuple:

[terminal id, parameter set, source module id, function name]

The terminal id, source module id, and function name are short, fixed length identifiers which are grouped together into a header called a "message descriptor." The parameter set contains variable length textual material and comprises the "text" of a message. In cases where the message is conveying large quantities of data, the textual part can be very long. However, intermodule message channels are likely to be of relatively low bandwidth. For performance reasons, then, it is convenient to send the text separately. In Figure 6 the two different types of paths, for message descriptors and for text, can be noted. A physical mechanism for safely transmitting text among modules is described in the next section. It will be assumed thereafter that the text is free from tampering. Further, no text can be transmitted without a proper corresponding message descriptor. The composition of the descriptor part is described below and the way in which these two parts are transmitted through the system is discussed in the next section. After that, it is sufficient to consider only the descriptor and how it is transmitted through the system.

The message descriptor is composed of three parts:

1) a message classification
2) a message identifier (or ID)
3) a message text address

The "message classification" is a numeric code (discussed in section) which identifies, among other things, the source module and function name of the 4-tuple. The "message ID" contains unchangeable message identification markings which associate the message with a specific terminal id. It can also contain information about user identification, job name, etc., and even a time stamp to indicate when the message was initiated. The "text address" is the memory address of the message text. As an example of a message descriptor and text, consider a Call DataBase (CDB) message as illustrated in Figure 5. The CDB message descriptor contains the classification code of 105 (explained later), the "message ID" that associates this CDB message with a particular user, and the text address of (or pointer to) the request tables constructed in the UAM memory.

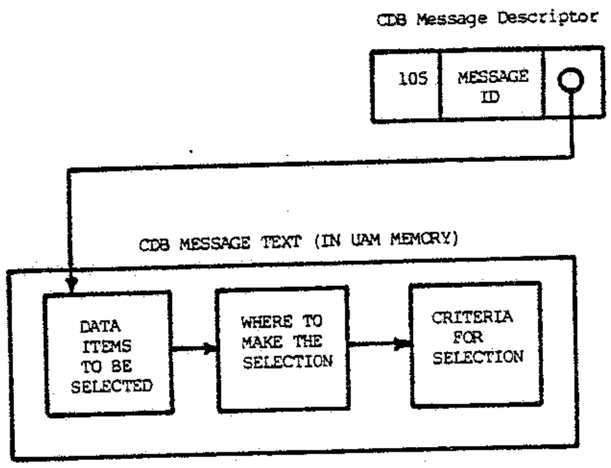

CDB Message Descriptor
105
MESSAGE ID
CDB MESSAGE TEXT (IN UAM MEMORY)
DATA ITEMS TO BE SELECTED
WHERE TO MAKE THE SELECTION
CRITERIA FOR SELECTION

Figure 5. Example message descriptor and text for CDB

6.4 Message Paths

Primary Memory Connections

A proposed basic (or minimal) architecture for MULTISAFE is shown in Figure 6. Although MULTISAFE can be implemented in many ways, including by virtual processes on a single hardware processor, its architecture is best described as a multiprocessor configuration composed of three separate processors which are connected to three separate primary random access memory blocks. This system organization follows the concepts outlined by Enslow [ENSLP77] for a multiport-memory organization with private memories. A multiport-memory is a primary memory

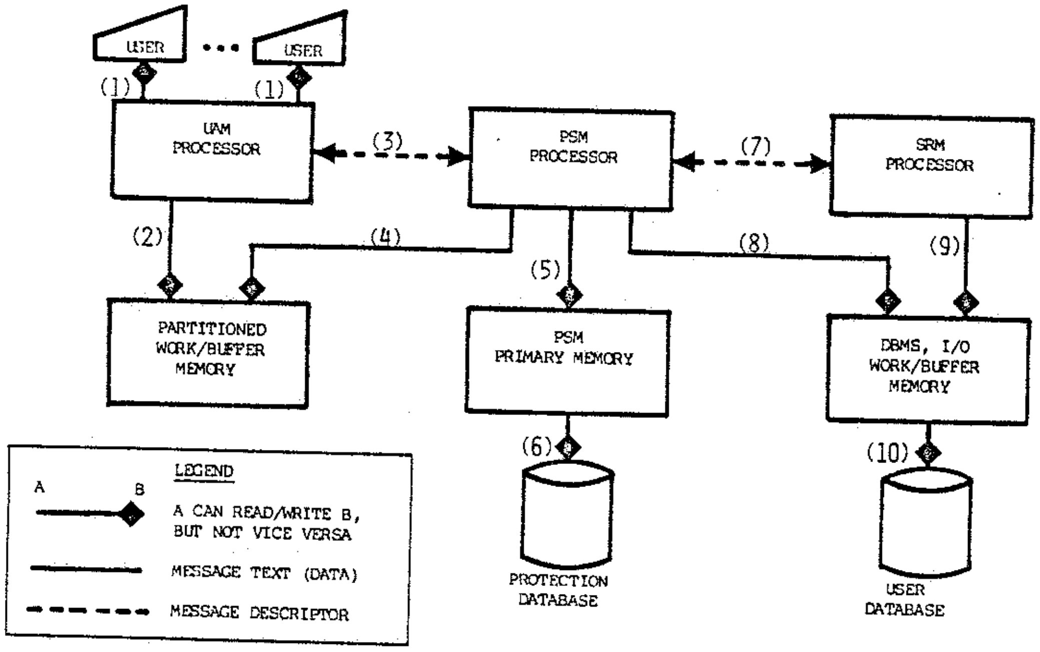


Figure 6. Message paths in MULTISAFE architecture

block with additional switching logic in its interface unit to allow access by more than one processor. The interface logic contains a priority arbitrator for resolving concurrent memory accesses. When a processor is connected to more than one memory, the ability to access any one memory block is the same except for priority waits.

A memory block can be made "private" by connecting only certain processors to it, thereby providing physical separation between, for example, the user's memory and the PSM and SRM memories. In a virtual processor implementation, the multiport and private memory features can be emulated by controlling the way in which address spaces are shared.

Efficient Flow of Text

Some implementation aspects are concerned mainly with performance. To illustrate, consider the flow of retrieved data among the modules. By expanding the multiport memory switching logic somewhat, it is possible to achieve logical transfer of data from one primary memory to another without having actual physical movement of data. For example, a portion of SRM memory can be "lent" to the PSM, logically transferring its contents in a hardware switched version of the way buffer pointers are exchanged within operating systems. Logically, the PSM passes judgement by making an access decision before relaying data from the SRM to the UAM and eventually to the user. The logical view, then, puts the PSM in the data path between the SRM and the UAM. An efficient implementation might accomplish the transfer directly from the SRM to the UAM under strict control of the PSM. It is the burden of the implementor to show conclusively that such a PSM-controlled switching mechanism is equivalent, in terms of security, to the logical view discussed earlier in this paper.

The PUSH/PULL Mechanism

The fixed format descriptor is sent between modules via an encapsulated data type (involving the 4-tuples, the pairs, and the queues of section 5.2). Its contents are set and checked by protected and verified procedures which are invoked parametrically. No user or user process can directly access message descriptors.

All message text is sent memory-to-memory within the private memory structure under the control of the PSM either by "PUSHing" (the PSM depositing the text in the UAM's or SRM's memory) or by "PULLing" (the PSM retrieving the text from the UAM's or SRM's memory). The PUSH/PULL mechanism is represented by the double-headed arrows in Figure 6 and protects the PSM memory from any kind (read or write) of access by either the UAM or SRM. The PUSH/PULL mechanism is implemented within the hardware switching logic of the private multiport memories.

The relationship between the PUSH/PULL mechanism and access decision binding times (see [HARTH77]) is important. For example, it might be possible that the text of a message from the UAM to the PSM be altered in the UAM, as the UAM is not completely verified and is also more open to inputs from other places in the system. However, no such alteration can occur after it has been PULLed into the PSM, since the PSM will be verified and no other module can modify its memory contents. Thus, delaying access checking until the message is safely in the PSM ensures that the checked form of the request is the form which is finally processed by the system, and the door to further tampering is closed.

6.5 Processing Steps

MULTISAFE functions in a multiuser environment. Each processor tries to remain busy. For the sake of fast response to legal requests, the basic philosophy is to assume that each incoming request is in fact a legal, authorized request--unless and until it is otherwise determined. The UAM and the SRM proceed to perform as many user oriented and DBMS functional operations as possible, independently from the PSM but without returning any information to the user or modifying the database in any way until they receive permission to do so from the PSM. The result is concurrency which is a combination of parallelism and pipelining.

It is now useful to trace the step by step processing of a single user request through MULTISAFE. The numbers in parentheses in each step correspond to those numbers in parentheses in Figure 6. A previous valid login is assumed to have been accomplished.

1) The user's request enters the UAM by (1) and is placed in the UAM memory through (2).

2) Next the UAM performs syntax analysis on the user's request and constructs request tables for the SRM.

3) When the PSM is ready to process a new query, it requests one from the UAM through (3). When step 2 above is completed, the UAM responds with a notification through (3)--which is passed on to the SRM through (7)--that a user's request is ready for processing. The message which

starts the SRM is a "CDB" (Call DataBase) call which is similar to "SIO" (Start I/O) supervisor macro call in an OS. At this point, the UAM enters a wait state (or begins processing another user's request) for the query results.

4) Next the PSM extracts through (4) a copy of the request tables in the UAM memory, and through (8) places these tabes into the SRM memory. The SRM begins database accesses.

5) Upon notification of the access request in step 3 above, the PSM retrieves from its database, through (6), the appropriate part of that user's authorization information--the security procedures and access conditions which apply to this request (predicates that determine access privileges). The PSM processor through (5) begins the enforcement process as specified by the user's authorization information. If a security procedure requires additional information from the user, such as a password, then the PSM sends a message through (3) to the UAM. The UAM interrogates the user and returns his/her response to the PSM through (3).

6) While the SRM is busy in step 4, the PSM examines the copy of the request table through (8) and the access conditions through (5) to determine the need for data-independent and/or data-dependent checking. First, the PSM performs the data-independent checks such as attribute name checking. Some of the things that can be checked are user ID, terminal number, time of day, and other status information. Some data definition dependent checking can be done here, also. For example, for a request in a query language like SEQUEL [CHAMD76], the selection and predicate domains (attributes) can also be checked at this time. If data items for data-dependent checks are required, the PSM informs the SRM through (7) that additional data items are to be retrieved for security checking. A list of these data items is constructed in the SRM memory through (8).

7) After the SRM has received notification through (7) from the PSM about data items for data-dependent checks, the SRM initiates the retrieval for these items, if additional retrieval is necessary.

8) Data retrieved by the SRM is placed in the SRM memory through (10), and the SRM processor performs any needed data manipulations on the retrieved data through (9).

9) When the SRM has prepared a set of data items (record or block), the PSM is notified through (7). The SRM continues the retrieval process by collecting the next set of data items in another buffer area. That is, the SRM processing returns to step 8 unless the CDB is satisfied, or unless the PSM orders the SRM to halt because of an unauthorized access attempt.

10) After the SRM has notified the PSM as in step 9 above, the data is pulled through (8) from the SRM memory and into the PSM memory through (5). The PSM examines the data through (5) and performs the data-dependent checks.

11) If access to the retrieved data is authorized, the PSM puts the data into the UAM memory through (4) and notifies the UAM of this action through (3). At the same time, if this completes the CDB, the next query is requested from the user. (For unauthorized access attempts, the PSM takes control and administers alarm and/or recovery procedures.) If this does not complete the CDB, the PSM returns to step 10 to get the next set of data from the SRM.

12) When the UAM has been notified by the PSM, as in step 11 above, it returns the results to the user through (1).

From the processing flow given above, two processing loops can be identified. One loop is in the SRM where steps 8, 9, and 10 are repeated for each set of data (block) that is retrieved for the user. The other loop is in the PSM where steps 10, 11, and 12 are repeated for each block of data that is retrieved. These two loops are being processed in parallel with each other.

6.6 Message Classification

A message is characterized by four attributes. These attributes are:

1) class
2) source
3) target
4) type

Messages are grouped into two classes--request and response. For each message there is a source (module), a target (module), and a message type. The source is the module which generates the message. The message target is the module which receives the message. The UAM, SRM, and PSM can each be a source and/or target module (depending on the other attributes). The user and authorizer have been subsumed into the UAM, since all messages from the user/authorizer are sent to only the UAM and all messages to the user/authorizer are from only the UAM. (This also eliminates any differences between interactive user initiated queries and calls to the database from host programs executing on behalf of a user.) The message type identifies the function being returned from or called. Authorization type messages are those messages which inquire about or modify the authorization information in the PSM. Information type messages request or return additional information needed by the PSM to make an access decision. The other types are more or less self explanatory. The range of each attribute is a finite set of values:

CLASS = {request, response}

SOURCE = {UAM, PSM, SRM}

TARGET = {UAM, PSM, SRM}

TYPE = {log, access request, authorization, information}

The set of all possible Message Classifications, MC, is given by the cartesian product of the above four sets:

$$MC = CLASS \times SOURCE \times TARGET \times TYPE$$

A specific classification is represented by the four-tuple

$$(c, s, t, p)$$

where $c \in CLASS$, $s \in SOURCE$, $t \in TARGET$, $p \in TYPE$.

The security constraints of the architecture are such that not every classification in MC is allowable in MULTISAFE. For example, a database user as a source is not allowed to send a message directly to the PSM as a target. There is a non-empty proper subset of MC which will be referred to as the secure message classification (SMC) set.

The four attributes are used to establish a hierarchy of secure message classifications. At the root of the structure is the collection of all secure message classifications. The next four levels are used to represent the four attributes--class, source, target, and type--respectively. Each terminal node in this structure is assigned a numeric code, called an SMC number, denoting the message classification.

Request Class

The class attribute partitions the set of message classifications, MC, into two equivalence classes (subtrees). These classes are for requests and responses. The request class contains those messages which are calls for data or information. These messages require a response from their receivers. The structure for classifying messages in the request class is a tree. The terminal nodes of this tree structure represent four tuples, the meaning for each of which is found in Table

TABLE I

REQUEST MESSAGES

SMC NO.	Source	Target	Type	Meaning
101	UAM	PSM	log	user wants to login (call which activates PSM_P)
102	UAM	PSM	access	database access request (return from USER_REQUEST call)
103	UAM	PSM	auth'n	authorization request (return from USER_REQUEST call)
104	PSM	UAM	info	additional information required for enforcement decision (e.g., USERID call)
105	PSM	SRM	access	SERVICE call (Call DataBase), pass user request to SRM
106	PSM	SRM	info	additional information required for enforcement decision

I. The first column of the table gives the SMC number which is used in each message as the "message classification." The first digit of the SMC number, which is the digit 1, identifies these messages as belonging to the request class. The last two digits are sequence numbers used to identify each specific request.

Response Class

Those messages in the response class are answers to the messages in the request class. The structure for classifying messages in the response class is a tree. The terminal nodes of this tree represent four-

TABLE II

RESPONSE MESSAGES

SMC NO.	Source	Target	Type	Meanings
201	PSM	UAM	login	login decision (and solicits acc. req. by USER_REQUEST call)
202	PSM	UAM	access	returns retrieved data (and solicits next acc. req. by USER_REQUEST call)
203	PSM	UAM	auth'n	authorization results or response
204	UAM	PSM	info	returns additional information (e.g., return to USERID)
205	SRM	PSM	access	return from SERVICE call (CDB)
206	SRM	PSM	info	returns additional information

tuples, the meaning for each of which is found in Table II. In the first column of the table is the SMC number which is used in each message as the classification code. The first digit of the SMC number, which is the digit 2, identifies these messages as belonging to the response class. The remaining two digits in the SMC number are sequence numbers used to identify each specific response.

The request and response trees are very similar. The main difference between the two trees is that the target of a given request has become the source of the corresponding response and vice versa. Every path from the root to a terminal node in the request tree has a corresponding path in the response tree where the source and target have been interchanged. This relationship implies that for every request in the SMC set there is a response in the SMC set. The last two digits of a response SMC number match those of the corresponding request SMC number. (This is not a security requirement, but a useful mnemonic device.) Notice also that the PSM is either the target or the source of every message, a necessary condition for all messages to go through the PSM.

6.7 Message Sequences

A message sequence is an ordered series of messages (calls and returns) among the modules of MULTISAFE, in response to a request from a user or an authorizer.

All message sequences are subject to two kinds of security checking:

1) checking specific to the request

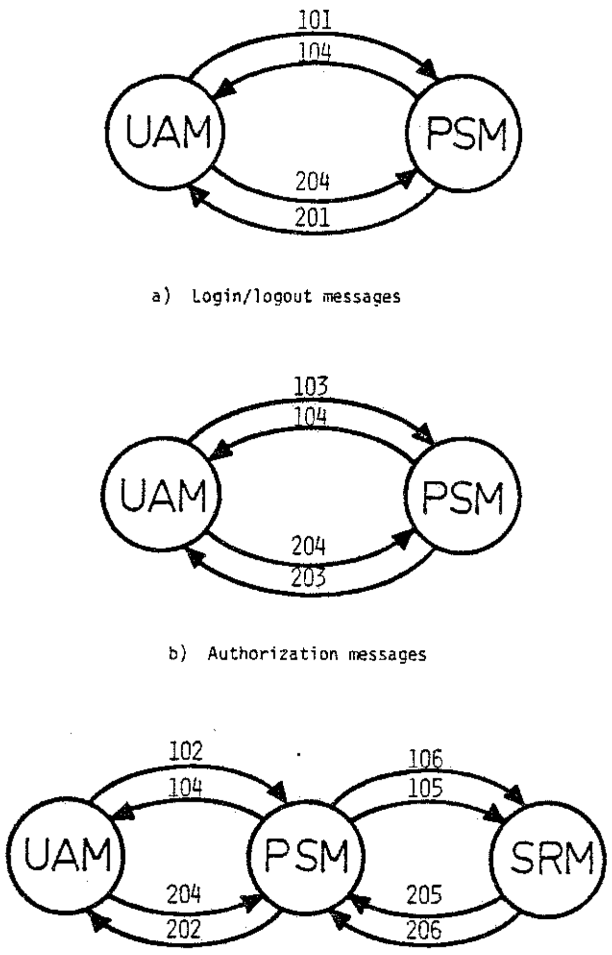
2) system occupancy checking

The specific check for a login request involves user identification , project name, etc. possibly augmented by password, authentication dialogues, etc. The specific check for a data request employs data-independent and data-dependent access conditions. System occupancy checks relate to overall permission to be an active user of the system, without regard to how it is being used. The system occupancy check is always, at a minimum, made in conjunction with login. 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 provides an (optional) additional binding time for these conditions.

Nesting and Subsequences

Within a message sequence every request message has a response. A request and its response form a <u>request/response pair</u>. It is possible for request/response pairs to be nested within other pairs. For example, a user's request to login may contain another pair of messages such as the request/response for a password before a login response is given. Even though nesting of message pairs is a key building block used in the construction of message sequences, not every message sequence is a perfect nesting of pairs. That is, message sequences can have non-nested adjacent pairs. Further, although requests and reponses can always be paired up, in practice several responses may pair up with the same request. This is because responses containing large amounts of data must be broken up and sent back a block at a time, each block being part of the response to the same request. But, logically, the blocks can be viewed as a single response. (Examples and explanation are given in the next section.)

The request/response pairs of a message sequence are illustrated in



a) Login/logout messages

b) Authorization messages

c) Database access messages

Figure 7. Illustration of request/response pairs

Figure 7. The arcs in the figures represent the messages. The direction of the arcs illustrate the flow of messages from source to target. The numbers on each arc correspond to the SMC number as given in Tables I and II. Request/response pairs are identified by the commonality of the last two digits of their SMC numbers. The nesting of some pairs is

illustrated by nested arcs. For example, in Figure 7a the pair (104, 204) is nested within the pair (101, 201). Such a nesting represents a request to login which requires a request/response for, say, a password before the login response is made. Figure 7c shows an example of another nesting--the pair (105, 205) in the pair (102, 202). Such a nesting within the message sequence (which does not result in nested arcs) illustrates how the database access request gets passed along to become a CDB, in the form of a request/response pair (105, 205).

Examples of Message Sequences

Presented below are some examples that illustrate the flow of message sequences through MULTISAFE. Accompanying these examples are figures that show the message direction, message type, and nesting of messages. Message direction is depicted by an arrow, each of which is identified by the message SMC number (see Tables I and II). Message nesting is depicted by loops. Some of these loops are formed with a dashed line that connects the request message with its corresponding response message when other pairs are embedded.

User Initiated Message Sequences

The first example, in Figure 8, illustrates a message sequence for the login message type. Using the SMC numbers, the order of the messages in this message sequence is as follows:

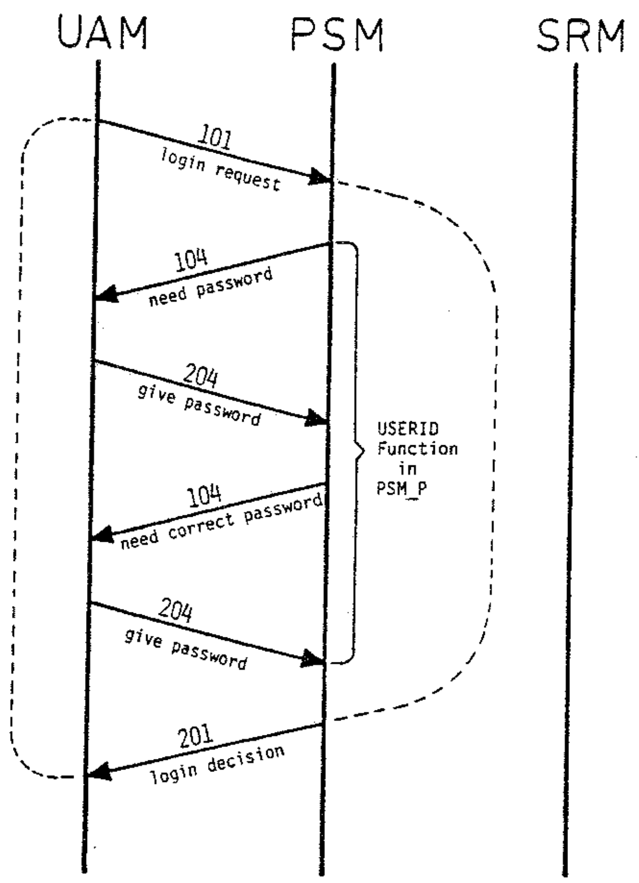


Figure 8. Login message sequence example

101, 104, 204, 104, 204, 201

In this message sequence, representing a login check, the first nested pair (104,204) is a request and a response for a password. The first password was incorrect, in this example, and the PSM had to make a second request for the password. Once the PSM has the correct password, the login check (which is the initial system occupancy check) is completed (at the point on the PSM line between the adjacent 204 and 201 messages) and the user is allowed onto the system.

If the second password had also been incorrect and the user was not allowed on the system, the message sequence in Figure 8 would not have changed. Only the message text for response 201 and the user status in the PSM would be changed. (The number of times a user can attempt to enter the correct password is within the policies of the authorizer and is not limited by MULTISAFE.)

Figure 9 illustrates the message sequence for query processing in which several blocks of data are retrieved and passed to the user. The message sequence begins with the UAM response to the PSM's USER_REQUEST call with a 102 message, asking for database access. The PSM copies the 102 text from the UAM's memory to its memory. At this point in time the binding of the database access request takes place (i.e., the access request text can no longer be modified by the user). Next, the PSM performs data independent security checking. This is done by the CHECK1 function in PSM_P. For this user request, it was necessary to reauthenticate the user's identity, using the (104, 204) messages. After clearing CHECK1, the PSM calls the SERVICE function, initiating the 105 message. The SRM retrieves the blocks of data from the database and returns them to the PSM for data-dependent checking (205). For this particular query the PSM needs additional information from the database before data-dependent checking is completed by the EXTRACT function. The message pair (106, 206) furnishes the PSM with the needed data. The

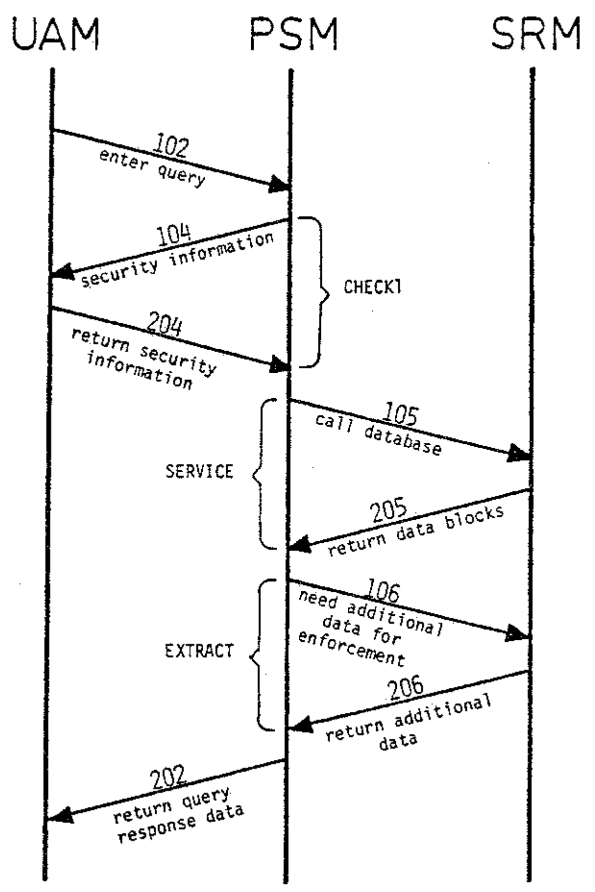


Figure 9. Query processing message sequence example

PSM completes its security checking and passes these blocks of data (202) to the UAM.

Sometimes blocks of data are not authorized. This can result in only certain blocks of data being passed on to the UAM and the user. In such a situation, given a partial enforcement policy [HARTH77], the

EXTRACT function deletes these unauthorized blocks and the unauthorized data ventures no further than the PSM.

If the volume of data retrieved from the database is large, multiple 205 messages may occur. Instead of waiting for all the data, SERVICE may return some of these blocks so that the EXTRACT function can be processing them while the SERVICE is retrieving more blocks.

It is possible for data access attempts to be denied by the system occupancy check. As an example of how the system occupancy check denies a data access, consider the case mentioned earlier in which the user is allowed to occupy the system only between 8:00 a.m. and 5:00 p.m. A potential penetrator logs in just before 5:00 p.m. in hopes of returning later, when the office is empty, to print a hard copy of an "eyes-only" file--an action which could readily be detected by co-workers during normal working hours. In this example, the plan could be defeated by the system occupancy check which can re-bind the login condition (just before the 105, 106, or 202 on the PSM line in Figure 9) at each request.

Authorizer Initiated Message Sequences

An authorizer is a user who is authorized to grant and revoke privileges for access to some of the system resources. Authorizers also set the conditions under which these accesses can take place.

Illustrated in Figure 10 is a message sequence for processing a request to display some authorization information (stored in the PSM

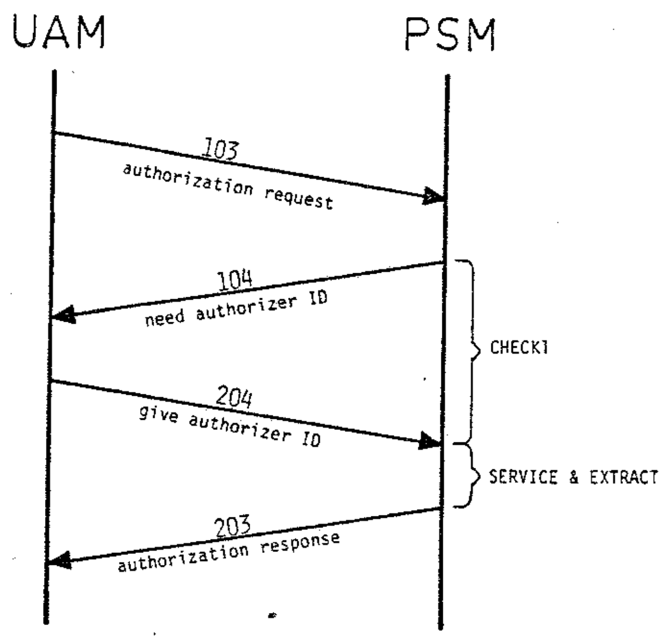


Figure 10. Authorization message sequence example

database as a result of previous authorizations). The UAM makes a request to the PSM (103) for the authorization information to be displayed. In the process of evaluating and checking the request, the PSM needs (in this example) some authentication information from the authorizer to verify his/her identity, thus the (104, 204) sequence. The authentication process is completed, along with any system occupancy checks, between the adjacent 204 and 203 messages on the PSM line. (Notice that the SERVICE and EXTRACT functions are used as in the case of database access; the main difference is that the PSM database is accessed instead of the SRM database.) If the enforcement procedure governing the authorizer's actions determines that the authorizer has the right to see the requested information, then the authorizer's display request is honored through the 203 message. Otherwise, the authorizer's display request is ignored and he is notified via the 203 message.

A message sequence for changing authorizations follows a pattern similar to that for the display.

7. FUTURE WORK

On-going and future work includes extensions, in several directions, of the work described in this paper. For example, an analysis of cost and performance is being undertaken. Simulation is being used to study both feasibility and performance. Petri nets are being applied to model the asynchronous concurrent processes and are being considered as a means to verify certain aspects of message security.

ACKNOWLEDGEMENTS

This paper is based, in part, on [TRUER79]. The authors gratefully acknowledge helpful discussions with Richard E. Nance and Lance J. Hoff-man.

REFERENCES

BANEJ78 Banerjee, Jayanta, Richard I. Baum, and David K. Hsiao, "Concepts and Capabilities of a Database Computer," ACM Trans. on Database Systems 3, 4 (December 1978), 347-384.

BISBR74 Bisbey, Richard L., II, and Popek, Gerald J., "Encapsulation: An Approach to Operating System Security," Proc. of the ACM Annual Conf. San Diego (November 1974), 666-675.

CANAR74 Canaday, R. H., Harrison, R. D., Ivie, E. L., Ryder, J. L., and Wher, L. A., "A Back-End Computer for Data Base Management," Comm. of the ACM 17, 10 (October 1974), 575-582.

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), 560-575.

COOKT75 Cook, Thomas J., "A Database Management System Design Philosophy," Proc. of the ACM SIGMOD International Conf. on Management of Data, (May 14-16, 1975), pp. 15-22.

COTTI77 Cotton, Ira W., and Paul Meissner, "Approaches to Controlling Personal Access to Computer Terminals," in Tutorial on Computer Security and Integrity, pub. EH 1124-8 by IEEE Computer Society (1977).

DENND76 Denning, Dorothy E., "A Lattice Model of Secure Information Flow," Comm. of the ACM 19, 5 (May 1976), pp. 236-243.

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), 76-96.

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

DOWND77 Downs, Deborah, and Popek, Gerald J., "A Kernel Design for a Secure Database Management System," Proc. of the 3rd International Conf. on Very Large Data Bases (1977).

EVANA74 Evans, Arthur, Jr., and William Kantrowitz, "A User Authentication Scheme Not Requiring Secrecy in the Computer," Comm. of the ACM, 17, 8 (August 1974), 437-442.

GOLDB79 Gold, B. D., R. R. Linde, R. J. Peeler, M. Shaefer, J. F. Scheid, and P. D. Ward, "A Security Retrofit of VM/370," Proc. of the AFIPS National Computer Conf. Vol. 48 (1979), 335-344.

GROHM76 Grohn, Michael, "A Model of a Protected Data Management System," ESD-TR-76-289, I. P. Sharp Associates Ltd., Ottawa, Canada (June 1976).

GUTTJ78 Guttag, John V., Ellis Horowitz, and David R. Musser, "The Design of Data Type Specifications," Chapter 4 of Current Trends in Programming Methodology, Volume IV: Data Structuring, (ed. R. T. Yeh), Prentice-Hall, Englewood Cliffs (1978).

HARRM76 Harrison, Michael A., Walter L. Ruzzo, and Jeffrey D. Ullman, "On Protection in Operating Systems," Comm. of the ACM 19, 8 (August 1976), 461-471.

HARTH76a Hartson, H. Rex, and Hsiao, David K., "A Semantic Model for Data Base Protection Languages," Proc. of the International Conf. on Very Large Data Bases Brussels (September 1976).

HARTH76b Hartson, H. Rex, and Hsiao, David K., "Full Protection Specifications in the Semantic Model for Database Protection Languages," Proc. of the Annual Conf. of the ACM Houston (October, 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), 349-356.

HOFFL70 Hoffman, Lance J., and W. F. Miller, "Getting a Personal Dossier from a Statistical Data Bank," Datamation 16, 5 (May 1970), 74-75.

HUNTJ79 Hunt, J. G., "Messages in Typed Languages," ACM SIGPLAN Notices 14, 1 (January 1979), 27-45.

KAMJB77 Kam, John B., Jeffrey D. Ullman, "A Model of Statistical Databases and Their Security," ACM Trans. on Database Systems 2, 1 (March 1977), 1-10.

KIRKG77a Kirkby, Gillian, and Michael Grohn, "On Specifying the Functional Design for a Protected DMS Tool," ESD-TR-77-140, I. P. Sharp Associates Ltd., Ottawa, Canada (April 1977).

KIRKG77b Kirkby, Gillian, and Michael Grohn, "Validation of the Protected DMS Specifications," ESD-TR-77-141, I. P. Sharp Associates Ltd., Ottawa, Canada (April 1977).

LANGT76 Lang, Tomas, Fernandez, Eduardo B., Summers, Rita C., "A System Architecture for Compile-Time Actions in Databases," IBM Los Angeles Scientific Center, Report No. G320-2682 (December 1976).

LAUEH78 Lauer, H. C., and R. M. Needham, "On the Duality of Operating System Structures," _Proc. of the Second International Symposium on Operating Systems_, IRIA (October 1978).

MCCAE79 McCauley, E. J., and P. J. Drongowski, "KSOS--The Design of a Secure Operating System," _Proc. of the AFIPS National Computer Conf._ Vol. 48 (1979), 345-353.

MILLJ76 Millen, Jonathan K., "Security Kernel Validation in Practice," _Comm. of the ACM_ 19, 5 (May 1976), 243-250.

NEUMP77 Neumann, Peter G., R. S. Boyer, R. J. Feiertag, K. N. Levitt, and L. Robinson, "A Provably Secure Operating System: The System, Its Applications, and Proofs," Final Report, Project 4332, SRI International, Menlo Park, Calif. 94025 (February 1977).

PETEJ79 Peterson, James L., "Notes on a Workshop on Distributed Computing," _ACM SIGOPS Operating Systems Review_ 13, 3 (July 1979), 18-27.

POPEG78 Popek, Gerald J., and David A. Farber, "A Model for Verification of Data Security in Operating Systems," _Comm. of the ACM_ 21, 9 (September 1978), 739-749.

POPEG79 Popek, Gerald J., et al., "UCLA Secure UNIX," _Proc. of the AFIPS National Computer Conf._, Vol. 48 (1979), 355-364.

PURDG74 Purdy, George B., "A High Security Log-in Procedure," _Comm. of the ACM_, 17, 8 (August 1974), 442-444.

ROBIL77 Robinson, L., K. N. Levitt, P. G. Neumann, and A. K. Saxena, "A Formal Methodology for the Design of Operating System Software," in R. T. Yeh (ed.), _Current Trends in Programming Methodology, Volume 1: Software Specifications and Design_, Prentice-Hall, Englewood Cliffs, N.J. (1977), 61-110.

SCHLJ75 Schlorer, J., "Identification and Retrieval of Personal Records from a Statistical Data Bank," Methods of Information in Medicine 14, 1 (January 1975), 7-13.

SCHWM79 Schwartz, Mayer D., Dorothy E. Denning, and Peter J. Denning, "Linear Queries in Statistical Databases," ACM Trans. on Database Systems 4, 2 (June 1979), 156-167.

TRUER79 Trueblood, Robert P., "Multiprocessor Architectures for Supporting Secure Database Management," Ph.D. Dissertation, Department of Computer Science Virginia Polytechnic Institute and State University, Blacksburg, VA (June 1979).

WALKB80 Walker, Bruce J., Richard A. Kemmerer, and Gerald J. Popek, "Specification and Verification of the UCLA Unix Security Kernel," Comm. of the ACM, 23, 2 (February 1980), 118-131.