Technical Report CS77009-R

A STUDY OF MULTIPROCESSOR ARCHITECTURES FOR SUPPORTING SECURE DATABASE MANAGEMENT

by

Robert P. Trueblood and H. Rex Hartson

December 1977

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

ABSTRACT

This is a work-in-progress report on multiprocessor architectures for supporting secure database management. The basic configuration contains three logical modules -- the user and applications module, the data storage and retrieval module, and the protection and security module. Each module resides on one or more processors that form a multiprocessor system called MULTISAFE. The background leading to the configuration and the functional characteristics of MULTISAFE are discussed. Some problems and questions for further research are also identified.

Keywords: Database Management Systems, Computer Architecture, Multiprocessor Systems, Security, Data Protection, Concurrent Processing, Access Control, Authorization, Enforcement

CR Categories: 6.22, 4.33, 3.7, 4.32

TABLE OF CONTENTS

1. INTRODUCTION 1

2. BACKGROUND 3

3. THE CONFIGURATION 8

A. PSM 9

B. UAM 13

C. SRM 15

D. Processing Flow 17

4. CONCLUSION 25

5. ACKNOWLEDGEMENTS 26

6. REFERENCES 27

INTRODUCTION

This is a work-in-progress report of a new proposed computer organization for supporting secure database management. In a conventional computer system, an increase in complexity of security mechanisms for greater precision and resolution (granularity) can degrade the performance of the system such that it is no longer cost-effective [CHASD73,HOFFL73]. In that type of system, processing is sequential and the user and system software share a common main storage. The database management system (DBMS) is implemented as a complex application "on top of" the operating system (OS), and security checking is performed by the DBMS and/or the OS. There are several problems with this type of architecture. For example, because of the common main storage, system software, security procedures, and other user data buffers can be read or altered by other software procedures. The database access mechanisms are usually more complex because they are processed through the operating system. The sequential nature of the whole system offers little aid in improving system performance. The new proposed architecture provides the next natural step in resolving some of these problems. To illustrate, for the sake of security and reliability, interprocessor communication is through more explicit and more controlled paths rather than through common memory access. By combining the concepts of multiprocessing, pipelining, and parallelism into the new system organization, improvements in system performance and security are possible.

In this report the basic configuration of a MULTIprocessor system for supporting Secure Authorization with Full Enforcement (MULTISAFE) for database management is presented. The general features of MULTISAFE are discussed along with its potential areas for future research. It is the intention of this report to state only the basic concepts without any details in order to document the early stages of development in the proposed architectures.

BACKGROUND

A DBMS can be functionally divided many ways for many purposes. Presented below are some of the ways in which DBMS's have been arranged into system configurations.

Canaday et al. [CANAR74] divided a DBMS into two levels -- the user and applications' level and the database management functions' level. With this view a "back-end" computer organization was developed. In the back-end organization non-DBMS processing is done by the host computer and DBMS processing is done by the back-end computer. The host computer transmits a user's request to the back-end computer where the DBMS functions are performed. The results are returned to the host for distribution to the user.

With an experimental model Canaday et al. showed that the back-end approach was economically feasible. Besides more throughput per unit cost, the system configuration enhances the ability for simultaneous data sharing and increases the system's reliability and security with back-up and recovery abilities.

The back-end computer resolved some of the access problems caused by having the database access mechanism on top of the OS as in the conventional system. Another feature of the back-end concept is that some concurrent processing is now possible. Thus, this provides a means for reducing the response time of the single conventional uniprocessor.

However, with respect to data security the back-end system is not much better than the conventional uniprocessor. That is, in back-end configurations security is processed either by the host or by the back-end computer. If security procedures are processed by the host, then they are susceptible to penetration by other software procedures being processed by the host. If security procedures are processed by the back-end computer, then system performance is degraded because the back-end computer (a uniprocessor) has to process the database access and security mechanisms sequentially.

A different partition was proposed by Bisbey and Popek [BISBR74]. The partition separated the operating system and security level from the user and application level. Bisbey and Popek proposed the implementation of their partition by extracting the operating system and the security procedures from the conventional computer and placing them on a minicomputer. This approach was called "encapsulation" and is related to the virtual machine concept. Under encapsulation the operating system and security processes became physically separated or isolated from the user and application procedures. Thus, it is easier to certify the security of the system from malicious attacks through application software. In addition, it is possible for each security category or level to be maintained as a separate operating system. This concept, called "compartmentalization," imposes an additional responsibility on the minicomputer whereby it must determine the security needs and initiate the loading of the operating system with the appropriate security level.

Encapsulation has some advantages. Security is isolated and easily certified. The operating systems are small and program verification is less difficult. Some limitations to encapsulation are that data security is very coarse (at the device and file level), and that file updates are infrequent. Thus, even though security is improved through encapsulation, it still does not resolve all the problems found in the conventional system.

Another approach to partitioning of data security and data management was taken by Downs and Popek [DOWND77]. In their approach the data security modules reside in the nucleus of the DBMS as a data protection kernel. This approach is similar to an OS kernel. The data management module, which maps logical data structures into physical data structures, is supported by a "data management kernel." All physical update and retrieval operations are performed by the data management kernel. For aid in updating, a second kernel, called the "kernel input module," is used for obtaining attribute-value pairs (the update data) from the user's update request because the data management module does not handle actual data. These attribute-value pairs from the kernel input module are matched and checked by the data management kernel with their physical data structures from the data management module. After security clearance, the data management kernel issues a physical I/O request that performs the actual update. For data retrieval, the kernel input module is not involved. The data management kernel validates the retrieval request from the data management module prior to issuing a physical database access.

The data protection kernel approach can improve the reliability of data protection since the kernel is separated from the data management module. However, the kernel approach, which performs data security only, provides no protection among concurrent DBMS users. This form of protection is left to the operating system. For example, there exists the possibility of one user tampering with another user's status since the data management module, which communicates error and status information to the users, has no protection mechanisms. System performance is degraded because of kernel overhead due to sequential processing for security checking.

With a future goal of implementing a relational DBMS on a distributed system of two computers Lien et al. [LIENY75] envision their system through three levels of abstraction: the logical data structure (relations), the physical data structure (binary search trees), and the physical machine. Their idea was to implement the primitive operations for manipulating binary search trees on both computers and to let the user's software be translated into these primitive operations. That is, both machines support the same virtual machine so that any user request can be processed by either machine. One advantage of this approach is that the two machines can be simultaneously processing different user's requests. Sometimes two requests can be processed in about the same time period as one request in the conventional system. However, besides the additional communication linkage between the two computers and the limitations imposed by the hierarchical tree structure, the approach suffers from having each request processed in the usual sequential manner of the conventional system.

In an effort to improve security Lang, Fernandez, and Summers [LANGT76] proposed a division of applications software (object programs) into three object modules:

1. the application object module,
2. the data interaction object module, and
3. the data control object module.

Due to the numerous calls among the three object modules, the overhead can degrade system performance under the conventional uniprocessor environment. For improving performance, Lang et al. suggested the use of an alternative architecture such as separate virtual machines, separate processors, or specialized architecture on which the object modules are executed.

Separate virtual machines have been explored by Cook [COOKT75]. Cook proposed a DBMS design philosophy based on the separation of structure representations. His idea is to separate logical (or user) structures from physical (or stored) structures and to separate both of these structures from the actual data. Relationships between structures are generated based upon the description of the user structures. The generated result is one's own "user machine" that is based on a tagged architecture which provides a set of data manipulation instructions. These instructions are interpreted in terms of one or more database machine operations. The final effect is several user machines operating independently with different structural views of the database tailored to each user's needs. Cook points out that it is possible to generate layers of user machines which easily support data protection by placing special user protection machines between the user machines and the database machine.

Even in this system organization, there still exists the linear processing sequence of operations for each user. The idea of layers does, however, support the virtual machine architectural alternative suggested by Lang et al.; but, these machines are simulated with a single physical machine which brings back some of the original problems found in conventional systems.

In the present work, some concepts from the system views and organizations outlined above are combined with several new ideas to form a new multiprocessor configuration. This configuration has the potential of resolving many of the problems of security and performance discussed above. The basic philosophy and concepts of MULTISAFE are discussed next.

THE CONFIGURATION

A computer system for a DBMS can be functionally divided 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)

The basic idea is to implement each module on one or more processors forming the multiprocessor system called MULTISAFE. These processors can be as small as microprocessors, or they can be as large as maxiprocessors. In a conventional uniprocessor environment these three modules function sequentially in an interleaved fashion. In MULTISAFE all three modules function in a semi-parallel fashion. That is, 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 1 illustrates the basic relationships among the three modules.

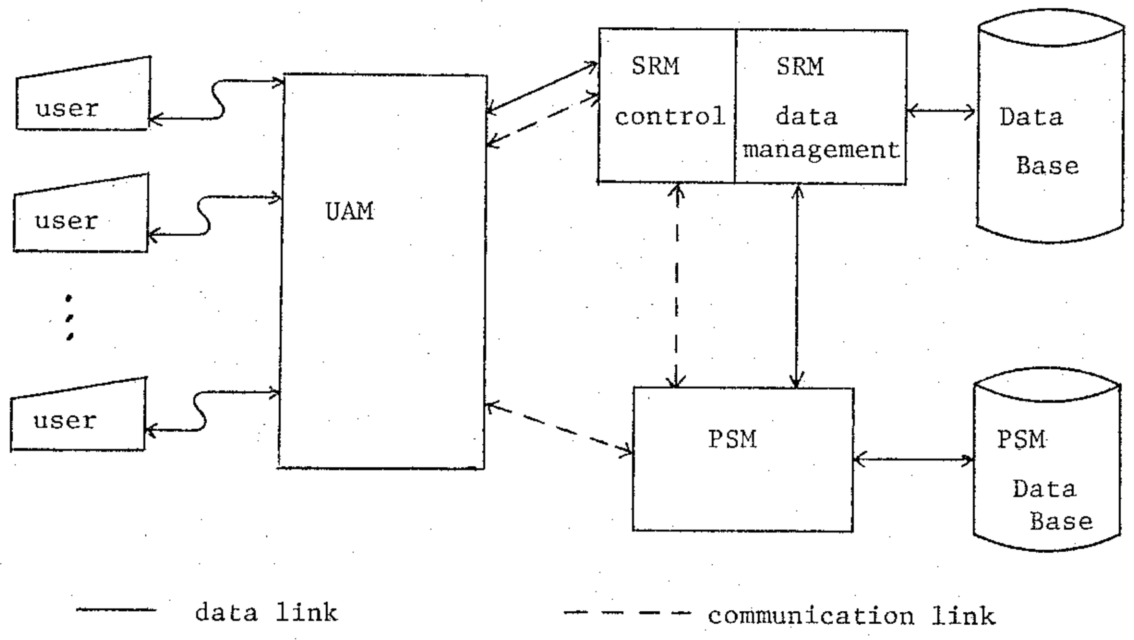

Figure 1. Relationships Among MULTISAFE Modules.

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. Using this assumption the UAM and the SRM perform as many DBMS functional operations as possible, independently from the PSM and 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 PSM operates concurrently with the UAM and the SRM. That is, the PSM is continuously performing security checks in parallel with the normal DBMS functional operations being performed by the UAM and SRM.

In order to gain a better understanding of the functional characteristics of MULTISAFE, each module is discussed individually. After the discussion of each module, the processing flow of MULTISAFE is explained.

PSM

The PSM is similar to the concepts of Bisbey and Popek, Downs and Popek, and Lang et al. in the sense that security mechanisms are encapsulated or isolated from other modules. The PSM differs from Bisbey and Popek's views because the PSM is dedicated to security checking and is not mixed in with other operating system functions (e.g., I/O handling). Furthermore, the PSM processor does not have to be reloaded for different security levels, and the PSM offers greater resolution and precision on data security checks which are needed for database systems. The PSM can serve as a DBMS kernel as Downs and Popek propose, but it performs only security checks and does not perform database I/O directly. Database I/O is done by the SRM. The PSM differs from the data control object module of Lang et al. in the sense that the PSM is more generalized and is not generated upon translation of an application program. The translation of an application program is performed by the UAM and pertinent information is passed to the PSM. Furthermore, changing the logical data model or implementing a completely different query language affects only the UAM. The PSM and the SRM are (more or less) independent of high-level database languages.

Each processor has a given set of basic functional operations to be performed concurrently with its companion processors. These basic functional operations for the PSM processor are as follows:

1. The PSM can interrupt and terminate further processing of a request by the UAM and/or SRM whenever a violation associated with that request has been determined.

2. When a new request has been accepted by the UAM, the PSM begins security checks by performing data-independent checking. Some of the things that can be checked are user ID, terminal number, time of day, and other status information. For a request in a query language like SEQUEL 2 [CHAMD76] the selection and predicate domains can also be checked at this time.

3. After the security checks in 2 above, the PSM determines whether or not data-dependent checking is needed. If not, the PSM informs the SRM simply to return the retrieved data (or the results of the request) to the UAM for disbursement to the user. If data-dependent checks are required, the PSM notifies the SRM that data-dependent checks are needed.

4. When a tuple (or record) or, in some cases, a group of tuples have been selected by the SRM in response to a request, the PSM is informed. Attribute-value pair checking is performed, if required. In some cases the PSM may need additional data to be retrieved before security checking is completed. Thus, the PSM may append (or modify) the original request prior to SRM processing. (This is similar to "AND"ing additional predicates to the user's query as Stonebraker and Wong have done in [STONM74]. This can help to minimize the amount of I/O by the SRM.)

5. After data-dependent checks are made, the PSM directs the SRM control to allow the results to flow from the SRM to the UAM. This is done

by informing the SRM and the UAM of the security check results.

It is the duty of the PSM to continuously perform the above functions. By maintaining status information along the way, the PSM can allow the request to continue or terminate. It is possible to allow authorized portions of a request to be processed and unauthorized portions to be denied -- particularly in data-dependent checking. The outcome in such cases depends on the level of resolution of enforcement [HARTH77].

From the functional operations described in 3 and 4 above, two types of data-dependent checking can be identified. One type will be termed internal data-dependent checking and the other, external data-dependent checking. For requests that require internal data-dependent checking, data checks are performed only on the domains or attributes specified in the request. This checking is internal to the request. For external data-dependent checking, data checks are required on domains or attributes that were not specified in the request. For example, consider the following SEQUEL 2 query.

```
SELECT employee-name, employee-salary
FROM employee-relation
WHERE department = 'administration'
```

Suppose that data dependent checks are to be made for the following policy. The user is not allowed to see any salary over $25,000 nor the salary of anyone who has a job title of manager. Since the attribute "employee-salary" was specified within the request, the check for salary over $25,000 is an internal data-dependent check. Since the attribute-value pair "job-title = 'manager'" was not specified in the request, the checking of job titles is an external data-dependent check.

The importance of the distinction between internal and external data-dependent attributes is that external data-dependent attributes may require additional I/O to be performed. That is, the PSM must modify the original request or issue its own request for data in order to perform security checks. If additional I/O is required, then system response to the request and system performance can be degraded. This is a problem area that needs further research.

The PSM is a software module that is processed on its own separate processor (or set of processors). This isolation protects the security system from malicious attacks through software and also provides the capability of physically relocating the processor remotely from the UAM and SRM processors.

The PSM can be viewed in terms of Hoffman's formulary model [HOFFL71]. In this view each user has a set of formularies (or procedures) to control database accesses. These procedures are called by the user and are processed prior to accessing the database. In MULTISAFE the database accesses and the access controls can be processed in a parallel manner or even in reverse order.

In view of Hartson's model [HARTH76a,HARTH76b] of protection languages, the PSM operates in two phases -- the authorization phase and the enforcement phase. In the authorization phase an authorizer uses a protection language for encoding policies (or access control rules), producing authorizations. Once these authorizations are processed and stored internally, the enforcement phase utilizes these authorizations along with system state information (including data contents) and the access request to produce an access decision in response to each database access request.

Some other functions that the PSM module might perform are history keeping, auditing, integrity checking, cryptographic processing, and backup/recovery processing. These functions might best be processed by a second PSM processor. Thus, the PSM can reside on dual processors, or might be distributed across several processors. However, in this paper, attention is focused on the PSM as residing on one processor that performs the authorization phase and the enforcement phase. The use of multiprocessors, history keeping, auditing, etc. are left as areas for future research. nother area of research is the implementation of the authorization phase and the enforcement phase.

An eventual requirement of this research is the ability to prove, or at least convincingly demonstrate, that security is guaranteed within the proposed architectures. Some of the potential security advantages of the PSM are listed below.

1. No data paths exist from the user to the database, except through the control of the PSM.

2. Authorization and enforcement are collected into one place which is physically separated from user modules and database access modules.

3. The UAM and SRM can function without the PSM; thus, those installations that do not require a PSM can detach it. (A PSM can be added at a later date.)

4. When the PSM is present, no user can detach it. A user cannot simulate detachment or simulate any of the PSM security functions.

UAM

The UAM acts as an interface between the user and the system, and the UAM does some computation for the user. The basic functional operations of the UAM are as follows:

1. Monitor all user input ports for an incoming request.

2. Lexically and syntactically analyze the input request and initiate further processing of the request. This consists (a) of informing the PSM that the request has arrived so that pertinent protection functions are activated and (b) of informing the SRM of the request so that information retrieval can begin.

3. Format and return to the user the results of his/her request.

All security and I/O routines are physically removed from the UAM and are completely isolated from the user. For example, suppose a user makes a database call. This call is passed simultaneously to both the PSM and SRM where it is analyzed and processed outside the user's environment. As opposed to I/O calls in the conventional system, this prevents the user from possibly tampering with the security, storage, and retrieval mechanisms.

The UAM differs from the other systems by the operations that are performed. For example, in the back-end system by 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 multiprogrammed processor such as an IBM 370. As a multiprogrammed system, some of the problems found in conventional systems (such as preventing unauthorized alteration of data in the common main storage) still exist. The second view is to have each user on a separate virtual machine similar to Cook's user machine which compartmentalizes the user's capabilities. However, the database is not in the virtual machines; thus, the data might still be subject to unauthorized tampering. A third 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. However, there may exist a need of another processor to coordinate the synchronization and communication of the user's intelligent terminal with the PSM and SRM. The problem of how to coordinate these terminals with the PSM and SRM is another area for research.

SRM

The primary task of the SRM is to perform database accesses for the UAM and PSM. The basic functional operations of the SRM are as follows:

1. Begin data access operations when UAM sends a request.
2. Notify the PSM when data has been retrieved.
3. After receiving security clearance from the PSM, return the results to the UAM, making any alteration to the data as specified by the PSM.
4. Continue processing 2 and 3 above until the request is completed or terminated.

Work in the area of developing a storage and retrieval processor (or device) for database systems has been done by Copeland et al. [COPEG73], by Ozkarahan et al. [OZKAE75,OZKAE77], and by Lin et al. [LINC76]. In general, their work concentrated on utilizing associative memory for database storage. Copeland et al. proposed a stand-alone, cellular system called CASSM which is constructed by attaching logic units to a head-per-track, segmented rotating storage device. CASSM was designed for supporting general database structures such as hierarchies. Ozkarahan et al. proposed a relational associative processor (RAP) that is a stand-alone system for supporting relational databases. RAP followed the idea of cellular architecture as found in CASSM but with more sophisticated logic for database accessing that improved processing time. Later, Lin et al. proposed a rotating associative storage device for relational databases, called RARES, in which search logic was attached to the fixed-heads of a rotating storage device for searching relational databases. RARES stores data across adjacent tracks whereas CASSM and RAP store data along adjacent tracks.

Since CASSM and RAP are stand-alone processors, they are likely candidates for supporting the SRM. However, more research is needed for studying the feasibility of using a CASSM or a RAP to support the SRM. By placing the database on a RARES storage device, the SRM processor becomes free from database searching. Thus, the SRM processor has more processing time for communication and data manipulation (or computation). Some other possibilities for supporting the SRM are specialized database machines such as those proposed by Baum and Hsiao [BAUMR76] and by Hsiao and Kannan [HSIAD77]. A possible area for research is to consider the pros and cons of using RARES, RAP, CASSM, and other database machines for supporting the SRM.

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.

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. In conventional systems all I/O operations are processed through the OS; that is, the DBMS is "on top of" the OS. In MULTISAFE all I/O operations are processed through the SRM; that is, the OS is on top of the DBMS, forcing <u>all</u> I/O (even simple file I/O) to be controlled by the protection mechanisms of the PSM. Another area for future research is to consider the advantages and disadvantages of having the OS on top of the DBMS as opposed to having the DBMS on top of the OS.

<u>Processing Flow</u>

MULTISAFE functions in a multiuser environment. Each processor tries to remain in the busy state by avoiding idle periods. For example, the UAM can begin analysis of another user request while the PSM and/or SRM are processing an earlier request. To gain insight into the processing flow, a description of the basic architecture of MULTISAFE is presented next. It is then followed by a step by step processing of a single user request through the system.

The basic (or minimal) multiprocessor architecture for MULTISAFE, as shown in Figure 2, is composed of three separate processors which are connected to three separate primary random access memory blocks (magnetic core or semiconductors, for example). The system organization follows the concepts outlined by Enslow [ENSLP77] for a multiport-memory organization with private memories. A

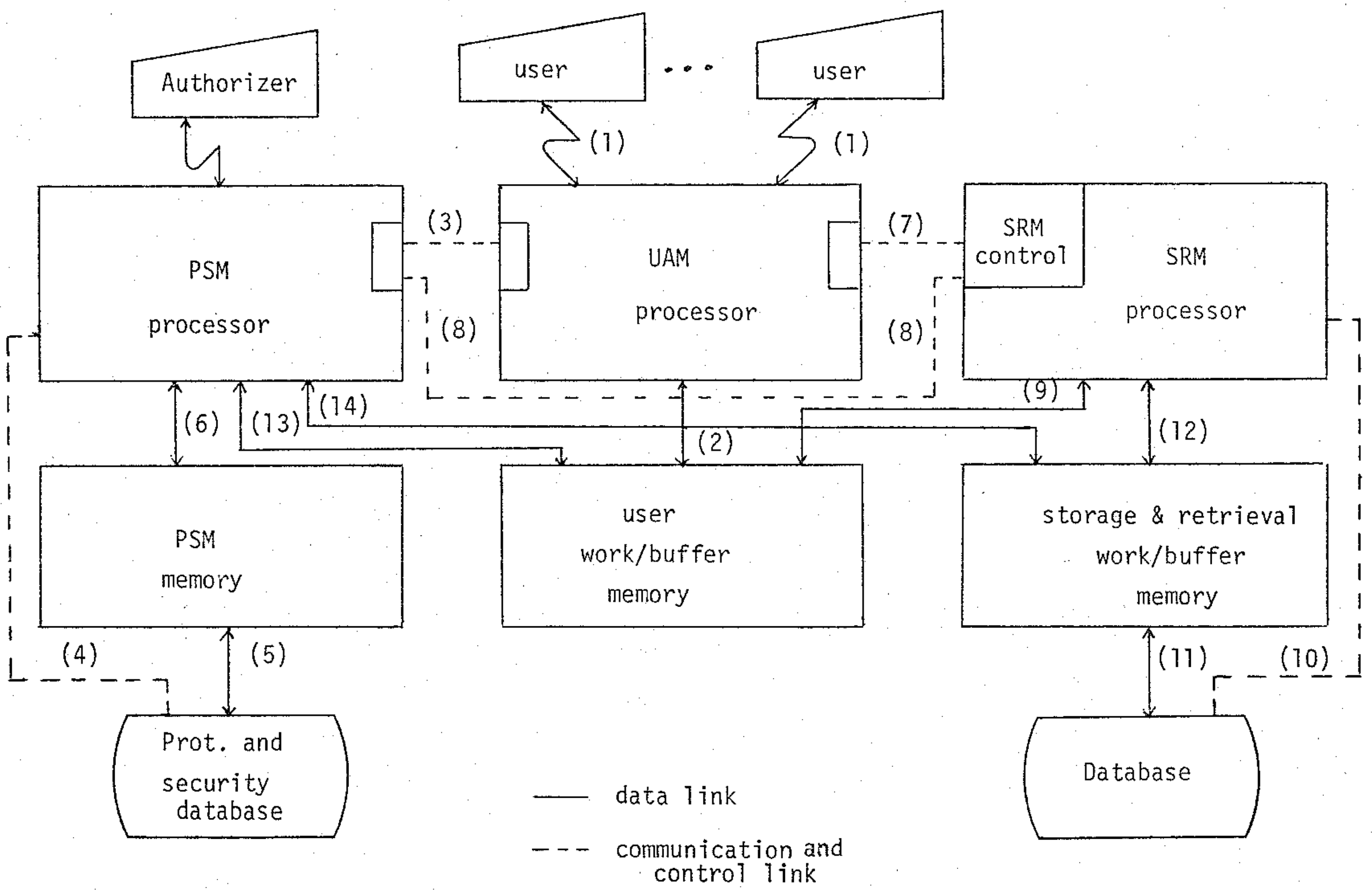

Figure 2. Basic architecture of MULTISAFE.

multiport-memory is a primary memory 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. Each memory port is uniquely designated a permanent priority during system construction. 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. The assignment of priorities can affect system performance and should be given careful consideration.

A memory can be made "private" by connecting only certain processors to it, thereby providing physical separation between the user's memory and the PSM and SRM memories, for example. This separation (or isolation) can significantly improve security because it is physically impossible for a user to access the PSM or the SRM memories. A unibus or a crossbar processor-memory connection does not provide this higher level of isolation which is a significant factor in improving security.

The multiport feature allows the PSM to access the UAM memory and the SRM memory. That is, the PSM has the capability at all times to examine the user's query and to control data manipulation operations. This capability provides a continuous monitoring effect. Also, the SRM has the ability to access the user memory so that retrieved and authorized information can be returned to the user.

Other than accessing information in the memories, the modules (or processors) require a more direct communication among themselves. This communication is necessary for coordinating and controlling the processing flow. Communication between the PSM and the SRM is a critical factor, especially for data-dependent security checks where communication is required on each record (tuple) processed. If this communication process is too slow due to operations such as passing and processing lengthy messages, then the performance of both the PSM and the SRM can be degraded. On the other hand, very short messages, such as single bit flags (or indicators), provide limited capabilities. Thus, the communication process needs careful planning and investigation to help preserve system efficiency.

An important part of the communication process is to assist the PSM in handling enforcement procedures. Each such procedure is composed of two parts: the interrupt mechanism and the recovery process. When the enforcement process detects an unauthorized request, the PSM interrupts the UAM and/or the SRM (depending on the nature of the enforcement) and administers the recovery process. Along with this interruption the PSM passes a message to the UAM which contains user identification information and a statement about the policy enforced. To assure security the communication process must be unavailable to the user, must be sophisticated enough to help the PSM administer enforcement, and must be able to satisfy all interprocessor needs. The communication process is another area that warrants further research.

Presented below is 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 2.

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

2. The UAM notifies the PSM through (3) that it has received a request from a given user on a given terminal.

3. Upon receiving the notification from the UAM, the PSM logs the system state (such as user id, terminal number, time of day, etc.). The PSM then places a request to its database

through (4) for the security procedures and access conditions (predicates that determine access privileges) for this user. These procedures are placed in the PSM memory through (5). The PSM processor through (6) performs the identification authentication as specified in the user's security procedure. If the security procedure requires additional information from the user, such as passwords, then the PSM sends a message through (3) to the UAM. The UAM interrogates the user and returns his response to the PSM through (3).

4. While the PSM is busy in step 3 above, the UAM is performing syntax analysis on the user's request and constructing request tables for the SRM and PSM.

5. When step 4 above is completed, the UAM notifies both the PSM through (3) and the SRM through (7) that the request is syntactically correct and that the request tables have been constructed. The message which starts the SRM is like a "CDB" (Call DataBase) supervisor call macro which is similar to "SIO" (Start I/O) in an OS. At this point, the UAM enters a wait state (or begins processing another user's request) for a response from the PSM and/or the SRM.

6. When step 3 above is completed, the PSM notifies the UAM through (3) and the SRM through (8) about any violations and the enforcement procedure takes control.

7. From step 5 above the SRM extracts through (9) the needed attributes for data retrieval from the request tables in the UAM memory. After extraction, the PSM is notified and the SRM begins preparation for database accesses.

8. When step 5 and 6 above have been completed and while the SRM is busy in step 7, the PSM examines the request table through (14) and the access conditions through (6) 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. If external data-dependent checks are required, the PSM informs the SRM through (8) that additional data items are to be retrieved for security checking. A list of these data items is constructed in the SRM memory through (14).

9. After the SRM has received notification through (8) from the PSM about external data-dependent checks, the SRM initiates retrieval for the items.

10. The SRM makes a database access request through (10) to the database. The retrieved data is placed in the SRM memory through (11), and the SRM processor performs any needed data manipulations on the retrieved data through (12).

11. When the SRM has prepared a set of data items, the PSM is notified through (8). 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 10 unless it has received notices from the PSM in step 13 that the current set of data is authorized. If so, the SRM processing follows step 14.

12. After the SRM has notified the PSM as in step 11 above, the PSM then examines the set of data through (14) and performs the data-dependent checks.

13. After step 12 above, the PSM notifies the SRM through (8) that the given set of data is authorized. The PSM returns to step 12 when the SRM has the next set of data ready for security checking. (For unauthorized data, the PSM enforcement procedure takes control and administers the recovery procedure.)

14. If the set of data is authorized, the SRM writes in the UAM memory through (9) the results of the user's request and notifies the UAM through (7). Then, the SRM processing proceeds to step 10 to collect the next set of data. If the next set of data has been collected, the SRM processing returns to step 11.

15. When the UAM has been notified as in step 14 above, it returns the results to the user.

From the step by step processing flow given above, two processing loops can be identified. One loop is in the SRM where steps 10, 11, and 14 are repeated for each set of data (or logical record) that is retrieved for the user. The other loop is in the PSM where steps 12 and 13 are repeated for each set of data (or logical record) that is retrieved. These two loops are being processed in parallel with each other.

Figure 3 illustrates the pipelining and parallelism of MULTISAFE for a user request that requires three records to be retrieved. The number above each time interval corresponds to the step number of the processing flow given above. Since no violations occur, step 6 is not processed.

From the above discussion on the functional operations of MULTISAFE, several areas for further research can be identified. Some of these areas have been identified earlier in this report but are included in this summary list

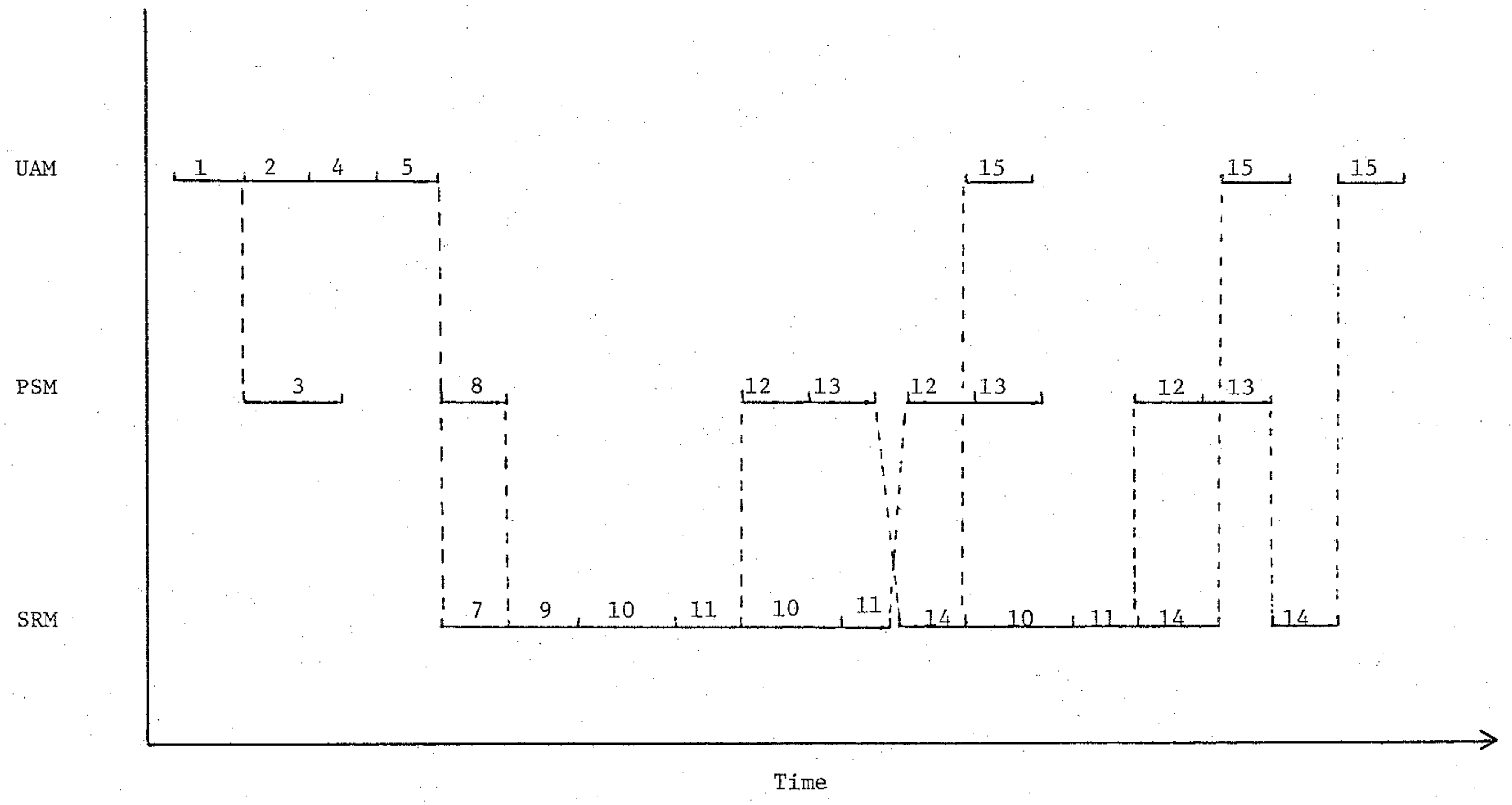

Figure 3. Illustration of MULTISAFE'S Pipelining and Parallelism That Retrieves Three Records for a Single User.

below. Each of these areas pose problems which can be answered in an experimental implementation of the proposed architecture.

1. Identify a spectrum of times and places in the flow of information where the PSM can bind access decisions.

2. Minimize the wait time due to interprocessor communication and synchronization.

3. Locate processor bottlenecks where one processor becomes overloaded with work, and begins to degrade system performance, while the other processors are waiting.

4. Verify the performance and protection advantages and disadvantages of MULTISAFE with respect to existing systems.

5. Develop a method for choosing processors (microprocessors, miniprocessors, maxiprocessors, or some combination) for the three modules.

6. Develop a technique to handle varying degrees of resolution of enforcement.

7. Identify new constructs needed in protection languages.

8. Develop a basis for analyzing performance and cost complexities -- particularly, measurements for protection enforcement.

9. Identify the system's performance with respect to query complexities such as external data-dependent security checks and large retrieval requests.

10. Prove, or demonstrate convincingly, that this approach leads to secure DBMS architectures.

11. Show that this multiprocessor approach gives better performance than the same hardware cost in a single, faster or more high-powered processor (i.e., justify breaking Grosch's Law).

12. Identify the relationships between the functional characteristics of an OS and the functional operations of MULTISAFE -- particularly, how the OS functions (i.e., process scheduling, memory management, etc.) are handled in a MULTISAFE environment.

CONCLUSION

In this report the functional characteristics of a multiprocessor system for supporting a secure DBMS has been presented. Some areas for further research have been identified. The authors' intend to continue the research and to answer as many of the posed questions as possible. They hope to design and implement parts (or all) of the basic configuration with microprocessors and to study the system's performance.

ACKNOWLEDGEMENTS

The authors wish to thank Professor Lance J. Hoffman, George Washington University, for his critical reading and helpful comments.

REFERENCES

BAUMR76 Baum, Richard I., and David K. Hsiao, "A Data Secure Computer Architecture," Compcon76 Digest of Papers, (February 24-26, 1976), pp. 113-117.

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

CANAR74 Canaday, R. H., R. D. Harrison, E. L. Ivie, J. L. Ryder, and L. A. Wehr, "A Back-end Computer for Data Base Management," CACM, Vol. 17, No. 10, (October 1974), pp. 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, Vol. 20, No. 6, (November 1976), pp. 560-575.

CHASD73 Chastain, Dennis R., "Security vs. Performance," Datamation, (November 1973), pp. 110-116.

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

COPEG73 Copeland, George P., Jr., G. J. Lipovski, and Stanley Y. W. Su, "The Architecture of CASSM: A Cellular System for Non-Numeric Processing," Proceedings of the First Annual Symposium on Computer Architecture, (December 9-11, 1973), pp. 121-128.

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

ENSLP77 Enslow, P. H., "Multiprocessor Organization -- A Survey," ACM Computing Surveys, Vol. 9, No. 1, (March 1977), pp. 103-129.

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

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

HARTH77 Hartson, H. Rex, "Dynamics of Database Protection Enforcement a Preliminary Study," Compsac77 (IEEE Proceedings on Computer Software and Applications Conference), Chicago, Illinois, (November 8-11, 1977), pp. 349-356.

HOFFL71 Hoffman, Lance J., "The Formulary Model for Flexible Privacy and Access Control," Proceedings of the Fall Joint Computer Conference, Vol. 39, (1971), pp. 587-601.

HOFFL73 Hoffman, Lance J., "IBM's Resource Security System (RSS)," in Security and Privacy in Computer Systems (ed. by Lance J. Hoffman), Melville Publishing Co., Los Angeles, Ca., (1973), pp. 379-404.

HSIAD77 Hsiao, David K., and Krishnamurthi Kannan, "The Architecture of a Database Computer -- A Summary," ACM SIGIR-SIGARCH-SIGMOD Third Workshop on Computer Architecture for Non-Numeric Processing, Syracuse, New York, (May 17-18, 1977), pp. 31-34.

LANGT76 Lang, T., E. B. Fernandez, and R. C. Summers, "A System Architecture for Compile-time Actions in Databases," IBM Los Angeles Scientific Center, Report No. G320-2682, (December 1976), pp. 1-26.

LIENY75 Lien, Y. E., C. E. Taylor, J. R. Driscoll, and M. L. Reynolds, "Binary Search Tree Complex -- Towards the Implementation of Relations," Proceedings of the International Conference on Very Large Data Bases, (Sponsored by ACM in Framingham, Massachusetts, U.S.A.), (September 22-24, 1975), pp. 540-542.

LINC76 Lin, C. S., D. C. P. Smith, and J. M. Smith, "The Design of a Rotating Associative Memory for Relational Database Applications," ACM Transactions on Database Systems, Vol. 1, No. 1, (March 1976), pp. 53-65.

OZKAE75 Ozkarahan, E. A., S. A. Schuster, and K. C. Smith, "RAP -- An Associative Processor for Data Base Management," AFIPS National Computer Conference, Vol. 44, (May 19-22, 1975), pp. 379-387.

OZKAE77 Ozkarahan, E. A., S. A. Schuster, and K. C. Sevcik, "Performance Evaluation of a Relational Associative Processor," ACM Transactions on Database Systems, Vol. 2, No. 2, (June 1977), pp. 175-195.

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