FILL IN - LABORATORY NAME - DATE

FORMULATIONS-- PROGRAM CONTROLLED PRIVACY IN LARGE DATA BASES

Lance J. Hoffman
Computation Group
Stanford Linear Accelerator Center
Stanford University, Stanford, California

ABSTRACT

This working paper describes a method of separating, in large computer data bases, the function of data addressing from the function of access control. This method is based on sets of access control procedures called Formulations. The decision on whether a user can read, write, update, etc., data is controlled by programs (not merely bits or tables of data) which are completely independent of the contents or location of raw data in the data base. This decision is made at data access time, not at file creation time as has generally been the case in the past. Indeed, the model presented does not make use of the concept of "files", though a specific interpretation of the model may do so. Access control is not restricted to the file level or the record level, although the model permits either of these. If desired, however, access can be controlled at arbitrarily lower levels, even at the bit level. Each element of raw data need appear only once, thus allowing considerable savings in memory and in maintenance effort over previous file-oriented systems.
TABLE OF CONTENTS

1. Introduction
2. General Concepts
3. Formularies -- What They Are
4. Use of Formularies
5. Summary
6. Acknowledgments

Appendix A. Algorithms for System Programs
Appendix B. Primitive Operations
1. Introduction

This working paper describes a method of separating, in large computer data bases, the function of data addressing from the function of access control. It presents a model in which the decision on whether a user can read, write, update, etc., data is controlled by programs (not merely bits or tables of data) which are completely independent of the contents or location of raw data in the data base. This decision is made at data access time, not at file creation time as has generally been the case in the past. Indeed, the model presented does not make use of the concept of "files", though a specific interpretation of the model may do so. Access control is not restricted to the file level or the record level, although the model permits either of these. If desired, however, access can be controlled at arbitrarily lower levels, even at the bit level. Each element of raw data need appear only once, thus allowing considerable savings in memory and in maintenance effort over previous file-oriented systems.

Specifically not considered in the model are privacy problems associated with communication lines, electromagnetic radiation monitoring, physical security, wiretapping, personnel, or administrative procedures. Cryptographic methods are not dealt with in any detail, though provision is made for inclusion of encrypting and decrypting operations in any particular interpretation of the model.

Specific interpretations of the model can be implemented on a standard, general-purpose computer with no special time-sharing or other hardware. The only provision is that all requests to access the data base must be guaranteed to pass through the data base system.
2. The Formulary Method of Access Control -- General

We now describe the "formulary" method of access control. This method separates the function of data addressing from the function of access control. Both functions proceed independently, as shown in the program of Appendix A. The decision on whether a user can perform operations such as read, write, lock, etc., on each datum is controlled by programs (not bits or tables) which are completely independent of the contents or location of raw data in the system. Access control is not restricted to the file level or the record level, although the method permits either of these. Access to individual related datums, which may have logical addresses very close to each other, can be controlled individually. For example, a salary figure might be released only without any identification of an employee.

The decision to grant or deny access is made at data request time, rather than at file make-up time, as has generally been the case in previous systems. This, together with the fact that the decision is made by a program (not by a scan of bits or a table), allows more flexible control of access and more efficient storage management. Data-dependent, terminal-dependent, time-dependent, and user response-dependent decisions can now be made dynamically at data request time, in contrast to the predetermined decisions made in previous systems, which are, in fact, subserved by the formulary method.

The basic idea behind the formulary method is that a user, a terminal, a formulary (defined below), and (sometimes) a data set, must be all linked together, or attached, in order for a user to perform information storage and retrieval operations. At the time the user requests use of the data base system, this linkage is effected, but only if the combination of user, terminal, formulary, and (sometimes) data set is allowed. The general linking process is described in Section 8.4, "The Attachment Process--The Method of Linking a Formulary to a User and Terminal".

IT IS ASSUMED THAT ENOUGH VIRTUAL ADDRESSING CAPACITY IS AVAILABLE TO HANDLE THE ENTIRE DATA BASE. VIRTUAL ADDRESSES ARE MAPPED INTO THE PHYSICAL CORE MEMORY LOCATIONS, DISC TRACKS, LOW-USAGE MAGNETIC TAPES, ETC., BY HARDWARE AND BY THE GET, PUT, GETANDLOCK, AND PUTANDLOCK MACROS (SEE APPENDIX B) FOR A PARTICULAR IMPLEMENTATION.
1. Formularies--What They Are

This section defines formularies, the sets of procedures which are the backbone of the access control method being presented. The next section gives examples of the use of formularies in a typical system.

A formulary is defined as a set of procedures, described below, which control access to information in a data base. These procedures are invoked whenever access to data is requested. They perform various functions in the storage and retrieval of information.

Different users will want different algorithms to carry out these functions. For example, some users will be using data which is inaccessible to others; the name of a particular data element may be specified in different ways by different users; some users will manipulate data structures--such as trees, lists, sparse files, ring structures, arrays, etc.--which are accessed by algorithms specifically designed for these structures. Depending on how he wishes to name, access, and control access to elements of the data base, each user will be attached to a formulary appropriate to his own needs.

The procedures of a formulary are called INTNAME, VIRTUAL, CONTROL, SCRAMBLE, and UNSCRAMBLE. In order to describe these procedures, we must first define some terms.
3.1. Definitions and Notation

The internal name of a datum is its logical address (with respect to the structure of the database). The internal name of a datum does not change during continuous system operation.

Examples:

A. A tree name such as S.7.1.2 which denotes field 2 of branch 7 of branch 5 in the data base.

B. Associative memory identifiers such as (14,273,34), where 14 represents the 14th attribute, 273 represents the 273rd object, and 34 represents the 34th value in a memory similar to the one described in (Rowen and Feldman 1968).

A natural language datum description describes a datum in terms of the user, as opposed to terms of the system.

Example:

(1) Stanford, Cowell, Payroll, Doe, Salary

The above natural language datum description represents the salary field of record Doe of file Payroll of Department File Cowell of Master File Stanford. The internal name corresponding to it might be (for example)

3.67.2.3.23

A User Control Block, or UCB, is space in primary (core) storage allocated during the attachment process (described in Section 4.4). It contains the user identification, terminal identification, and the virtual addresses of the INITNAME, VIRTUAL, CONTROL, SCHAMBLE, and UNSCHAMBLE procedures of the formulary the user is linked to. A data set name may also be included, for efficiency reasons.

The virtual addresses are kept in primary storage in the UCB since a formulary, once linked to a user and terminal, will probably be (oft-)used very shortly. The first reference to any of these addresses (indirectly through the UCB) will trigger some appropriate action (e.g., a page fault on some computers) to get the proper program into primary storage (if it is not there already). It will then presumably stay there as long as it is useful enough to merit keeping in high-speed memory. The virtual addresses of procedures of a formulary cannot change while they are contained in any UCB. This constraint is easy to enforce using the CONTROL procedure (see Section 1.2.1) which controls operations on any datums, including formularies.
In many instances, the user will be performing many operations on the same data set. When this is the case, it is both unnatural and inconvenient to (re-)specify the data set on each operation. The system eliminates the need for the user to do this by accepting a data set name once, during the attachment process, and saving it in the HCB. As each request to access data is received by the system, D is concatenated with the partial natural language data description of the request to form the complete natural language data description, which is then sent to INTNAME to be mapped into an internal name.
3.2. Procedures of a Formulary

In this section, we describe the procedures of a formulary. These procedures determine the accessibility, addressing, structure, and interrelationships of data in the data base dynamically, at data request time. They can be arbitrarily complex. In contrast, earlier systems usually made only table-driven static determinations, pre-specified at file make-up time. By use of the formulary method, these advantages are gained:

1. flexibility and changeability of data base organization to reflect current needs

2. capability to perform access control at request time as well as at file make-up time

3. more efficient use of storage

Each procedure of a formulary should, if possible, run from [effective] execute-only memory, which is alterable only under administrative control. The integrity of the system depends upon the integrity of the formularies and therefore the procedures of all formularies should be written by "system" programmers and audited for program errors, hidden "trap doors", etc., before being inserted into the (effective) execute-only memory under administrative control. Failure to do this may result in the compromising of sensitive data, since an unscrupulous programmer of a formulary could cause the formulary to "leak" sensitive information to himself or to his agents.

If a user who is not a "system" programmer does not wish to trust the confidentially of his data to a procedure of a formulary written by someone else (a "system" programmer), he may construct his own procedures for his own formulary. The system will permit this formulary only to access data in specific data sets chosen by that user, and thus insure that a user-written procedure does not allow compromise of the data of other users. The system can do this since all formulary building must be done using the program FORMULARY BUILDER.

Each formulary has five procedures: INNAME, VIRTUAL, CONTROL, SCRAMBLE, and UNSCRAMBLE. We now describe each of these in turn.
3.2.1. INTNAME

INTNAME translates a natural language datum description into an internal name, providing a bridge between the user's representation of the data base and the system's representation. INTNAME is an integer procedure with two parameters, FIELD and ANS. The natural language datum description is given in FIELD. The internal name is left in ANS. INTNAME will nearly always be table-driven. It returns

1 on normal exit

2 if it cannot translate FIELD (probably because of an unknown field name given it on input)
3.2.2. VIRTUAL

VIRTUAL translates an internal name into the virtual address of the corresponding datum. VIRTUAL is an integer procedure with three parameters, INAME, ANS, and OTHER. INAME contains the internal name to be translated. The virtual address is returned in ANS. OTHER contains sometimes valuable "other information" whose use is described fully in Section 3.2.3, "Control". VIRTUAL returns

1 on normal exit
2 on error exit

REMEMBER, IT IS ASSUMED THAT ENOUGH VIRTUAL ADDRESSING CAPACITY IS AVAILABLE TO HANDLE THE ENTIRE DATA BASE. VIRTUAL ADDRESSES ARE MAPPED INTO THE PHYSICAL CORE MEMORY LOCATIONS, DISC TRACKS, LOW-USAGE MAGNETIC TAPES, ETC., BY HARDWARE AND BY THE GET, PUT, GETANDLOCK, AND PUTANDLOCK MACROS.
3.2.3. CONTROL

CONTROL decides whether a user at his terminal is allowed to perform the operation he requests (READ, WRITE, LOCK, etc.) on the particular datum he has specified. CONTROL may consider and/or converse with the requesting user and/or terminal in order to arrive at a decision. CONTROL also checks for correct data set in the case of a formulary built by a person who is not a "system" programmer (see Section 3.2).

CONTROL is a procedure with two input parameters which returns a list as its output. The two input parameters are:

1. the internal name of the datum
2. the operation the user desires to perform

The output list has two elements. The first element is:

1. if access is allowed, and
2. if access is not allowed.

The second element contains "other information".

CONTROL returns "other information" as well as a "yes or no answer" for the following reason. In some specific systems, data elements may contain their own access control information. Consider two examples:

Example 1:

```
| DATUM | B | W | 30 bits of actual data |
```

If bit B is on, DATUM is readable.
If bit W is on, DATUM is writable.

Example 2:

```
| SALARY | 25,000 |
```

Reading or writing of salaries over $25,000 is not to be done in any case. CONTROL must inspect the SALARY cell before it can do further capability checking and eventually return 1 or 0 as the first element of its output list.
In systems of this type, CONTROL might often duplicate VIRTUAL's function of transforming the internal name of a datum into that datum's virtual address. To achieve greater efficiency, therefore, CONTROL can (when appropriate) return the datum's virtual address as "other information". VIRTUAL, which is called after CONTROL (see ACCESS in Appendix A), can then examine this "other information". If a virtual address has been put there by CONTROL, VIRTUAL will not duplicate the possibly laborious determination of the datum's virtual address, since this has already been done. VIRTUAL will merely pluck the address out of the "other information" and pass it back.
3.2.4. SCRAMBLE

SCRAMBLE is a scrambling procedure which transforms raw data into encrypted form. (In some specific systems, SCRAMBLE may of course be null.) SCRAMBLE has two input parameters:

1) the virtual address of the datum to be scrambled

2) the length of the datum to be scrambled

SCRAMBLE returns a three-element list, the elements of which are

1) a completion code (1 if normal completion)

2) the virtual address of the scrambled datum

3) the length of the scrambled datum
3.2.5. UNSCRAMBLE

UNSCRAMBLE is an unscrambling procedure which transforms encrypted data into raw form. (In some specific systems, UNSCRAMBLE may of course be null.) UNSCRAMBLE has two input parameters:

1. the virtual address of the datum to be unscrambled
2. the length of the datum to be unscrambled

UNSCRAMBLE returns a three-element list, the elements of which are

1. a completion code (1 if normal completion)
2. the virtual address of the unscrambled datum
3. the length of the unscrambled datum
3.3. Simultaneous Use of One Formulary by Multiple Users

Note that the same formulary can be used simultaneously by several different users with different access permissions. This is possible because access control is determined by the CONTROL procedure of the attached formulary. This procedure can grant different privileges to different users.

Note also that two formularies can contain the same CONTROL procedure but different INTNAME procedures, thus allowing two different users to use the same data but different natural language descriptions. For example, two formularies might be identical except for the fact that the INTNAME program of the first translated English "field name" into internal name, while the INTNAME program of the second translated French field name into internal name. This ability to use partially or wholly disjoint natural language descriptions for different users is not available currently in any system the author is aware of, though some systems go as far as providing "synonyms tables" (Jones 1968).

Different INTNAME programs also allow concealment of the fact that certain information is even in a data base, as illustrated in Figure 1.
Formulary of User 1 is: \{INNAME, VIRTUAL, CONTROL, SCRAMBLE, UNSCRAMBLE\}

Formulary of User 2 is: \{INNAME, VIRTUAL, CONTROL, SCRAMBLE, UNSCRAMBLE\}

**USER 1**  
WHAT DATA WOULD YOU LIKE TO SEND?  
salary of robert d. jones
YOU ARE NOT PERMITTED READ ACCESS TO  
THE SALARY FIELD.

CONTROL returned a list,  
the first element of which  
was 2, causing this reply  
to be given to the user.

**USER 2**  
WHAT DATA WOULD YOU LIKE TO SEND?  
salary of robert d. jones
NO FIELD NAMED SALARY

INNAME returned a value of 2, causing this  
reply to be given to the user.

---

**Figure 1.** Concealment of the Fact that a Data Base  
Contains Certain Information
3.4. Subdivision of Data Base into Files Not Required

Note that while the concept of a data set (or a "file") MAY be used, the formula method DOES NOT REQUIRE THIS. This represents a significant departure from previous large-scale data base systems which were nearly all organized with "files" as their major subdivisions. Under the formula scheme, access to information in a data set is NOT governed by the data set name. Rather, it is governed by the CONTROL program of the attached formula. Similarly, addressing of data in a data set is governed by the VIRTUAL program and not by the data set name. The data set name, while often useful (for reasons stated in Section 4.1) is not necessary for the system's operation.
3.5. Concurrent Requests to Access Data

The problem of two or more concurrent requests for data access necessitates a mechanism to ensure that a piece of data may be read and then written with no intervening operations on that data element. This problem has been discussed, and solutions proposed, in (Dijkstra 1965) and (Hsiao 1968) for the purposes of our model, data which must be read and then written with no intervening operations on that data is locked and unlocked at data request time in a manner similar to Hsiao's "blocking" (Hsiao 1968), using a mechanism known as the LOCKSTACK.

3.5.1. The LOCKSTACK--Separability of Blocking and Access Control Functions

The locking and unlocking of data to control simultaneous updating is an entirely separate function from the access control function. Access control takes into account privacy considerations only. Locking and unlocking are handled by a separate mechanism, the LOCKSTACK. The LOCKSTACK is a list of doublets maintained by the system and manipulated by the GET, PUT, GETANDLOCK, PUTANDLOCK, UNLOCKREAD, and UNLOCKWRITE primitive operations. Each doublet contains (1) the internal name of a current item, and (2) U, the identification of the user who locked it. No datum represented by a doublet on the LOCKSTACK can be accessed until that doublet is removed from the LOCKSTACK.
4. Use of Formularies

We now examine the use of formularies in more detail. A time-sharing system with teletype-like user terminals serves as the framework for our examples. Whenever specific interactive programs are used as examples, the conventions given in Table 1 are followed.
User types in lower case.

System (or user program) types in upper case.

Output messages from time-sharing systems are preceded by an exclamation point (!).

The symbol # is used to indicate user depression of the (nonprinting) carriage return and line feed key which terminates user messages and causes them to be sent to the computer.

<BR> indicates a "break character" which, when depressed by the user, causes the output message currently being printed at the user's console to be terminated.

**TABLE 1. CONVENTIONS FOR DESCRIPTION OF USER-SYSTEM INTERACTIONS**
4.1. Building a Formulary

Before a formulary can be attached to a user, terminal, and (possibly) data set, the procedures it contains must be specified. This is done using the system program FORMULARYBUILDER. FORMULARYBUILDER converses with the user who is building a formulary to learn what these procedures are, and then retrieves them from the system library and enters them as a set into a formulary which the user names.

In most cases, the user of FORMULARYBUILDER will be a systems programmer empowered by administrative action to build formularies for the use of all. If this is not the case, FORMULARYBUILDER will require a list of data sets from the (non-system) programmer. Only the information in these data sets will be available to someone using the formulary being built, and then only when other controls (enforced by the CONTROL program) allow.

The general block diagram of FORMULARYBUILDER is given in Figure 2. The specifics of FORMULARYBUILDER depend on the particular system. An example of the workings of FORMULARYBUILDER for a specific system is given in Fig. 3.

* An extension to FORMULARYBUILDER which would allow a user to grant capabilities to other users, thus to grant capabilities to still other users, etc., has been proposed by Victor Lesser and will be investigated further in the future.
Fig. 2. General Block Diagram of FORMULARYBUILDERS Program
WHAT PROGRAM DO YOU WISH TO RUN?
Formularybuilder
EXECUTION...
*** FORMULARY BUILDING PROGRAM ***

(a) YOU WANT INSTRUCTIONS ON THE USE OF THE FORMULARY BUILDING
PROGRAM. YES
FORMULARYBUILDER ALLOWS YOU TO DEFINE, OR BUILD, A FORMULARY
FOR USE AT A LATER TIME. YOU MUST SPECIFY FIVE PROCEDURES
TO PERFORM ACCESS CONTROL AND ADDRESSING FUNCTIONS. WHEN I
ASK YOU, PLEASE SPECIFY EACH PROCEDURE BY TYPING IN ITS
NAME (YOU MUST HAVE ALREADY PUT IT ON THE PROCEDURE LIBRARY.)

DO YOU UNDERSTAND THESE INSTRUCTIONS?
YES

ALL RIGHT. PLEASE SPECIFY THE INTNAME PROGRAM.
public:French,diagnosis,inventory
THANK YOU. PLEASE SPECIFY THE VIRTUAL PROGRAM.
public:standard,virtual
THANK YOU. PLEASE SPECIFY THE CONTROL PROGRAM.
private:cowell,control
THANK YOU. PLEASE SPECIFY THE SCRAMBLE PROGRAM.
private:cowell,scrambler
THANK YOU. PLEASE SPECIFY THE UNSCRAMBLE PROGRAM.
private:cowell,unscrambler
THANK YOU. WHAT DO YOU WISH TO NAME THIS FORMULARY?
health
IS IT TO BE PUBLIC, OR FOR YOUR OWN USE ONLY?
public
ARE YOU A SYSTEM PROGRAMMER AUTHORIZED TO BUILD PUBLIC
FORMULARY?

...YOUR HELP IS INDISPENSABLE, C. BARRAGE, CHIEF CUP. <<<<
...PUBLIC DEFENDS TO AUTHENTICATION CUPPER:
C547 -- T 0694 

...AUTHENTICATION VALIDATED. YOU HAVE DEFINED THE
HEALTH FORMULARY WITH THE FOLLOWING PROCEDURES:

INTNAME:  PUBLIC,STANDARD,INTNAME
VIRTUAL:  PUBLIC,STANDARD,VIRTUAL
CONTROL:  PRIVATE,COWELL,CONTROL
SCRAMBLE: PRIVATE,COWELL,SCRAMBLER
UNSCRAMBLE: PRIVATE,COWELL,UNSCRAMBLER

ANY OTHER INPUT DO NO
FORMULARY HEALTH HAS BEEN ENTERED INTO THE SYSTEM.

END OF FORMULARY BUILDING PROGRAM

WHAT PROGRAM DO YOU WISH TO RUN
FORMULARYBUILDER, in order to verify that the user is indeed a system programmer entitled to build public formularies, checks the user identification U (from the UCB) against its list of authorized system programmers.

Then, to double check, it requires the user to pass an authenticity check. In this case, the system supplies a pseudo-random four digit number and the user is supposed to answer with the fourth digit, then the second, then 1 + the first digit (mod 10). A check of this type is necessary to insure that people who are not system programmers do not make available for public use CONTROL programs that could destroy or compromise the sensitive data of other users (see Section 3.2).

Figure 3. Operation of the FORMULARYBUILDER Program in a Specific System
4.2. The ACCESS Program

The ACCESS program retrieves information from and stores information into the data base. It uses the INTNAME, VIRTUAL, CONTROL, UNSCRAMBLE, and SCRAMBLE procedures specified in the SCM to carry out these functions. In order for any information storage or retrieval operation to be carried out, the ACCESS program must be invoked. ACCESS has the following parameters:

1. Natural language datum description

2. A cell which either contains or will contain the value of the datum specified by (1)

3. Primitive operation to perform -- READ, WRITE, READANDLOCK, WRITEANDLOCK, UNLOCKREAD, or UNLOCKWRITE.

The primitive operations are specified and an algorithm for the ACCESS program is given in Appendix A, "Algorithms for System Programs".

The user communicates only indirectly with ACCESS. The bridge between the system-oriented ACCESS program and the application-oriented user is provided by the conversational storage and retrieval program, TALK (see Figure 4).
Figure 4. User/Data Base Interface
4.3. TALK, The Conversational Storage and Retrieval Program

To access a datum, the user must effectively call upon TALK, the Conversational Storage and Retrieval Program. TALK converses with the user (or the user's program) to obtain (1) a natural language datum description and (2) the operation the user wishes to perform. It then calls the ACCESS program to perform the operation desired. Depending on the particular system, the user may explicitly specify none, one, or both of the above two parameters which TALK sends off to the ACCESS program. Those he does not explicitly specify will be implicitly given values by TALK.

A system may have several TALK programs, depending on the data base structure and on the user's level of expertise. For an experienced user, a bare-bones program which reads parameters from the terminal and passes them along to the access program is sufficient; the novice user will require a much more elaborate TALK program (such as that described in (Kellogg 1968)) to build a bridge between himself and the ACCESS program.

The first task of TALK is to attach a formulary to the user and the terminal. This is discussed in the next section.
4.4. The Attachment Process—The Method of Linking a Formulary to a User and Terminal

In order to allow information storage and retrieval operations on the data base to take place, a user, a terminal, and a formulary which has been previously built using FORMULARYBUILDER must be linked together. A data set may or may not also be linked with the user, terminal, and formulary at this time. This linking, or attaching, process is done in the following manner.

The user initially is given the default UCB by TALK. This default UCB contains a user identification U and a terminal identification T which TALK has obtained either directly from the timesharing system in which the information storage and retrieval subsystem is embedded, or from conversations with the user and/or with the terminal. TALK, in addition to storing U and T in the default UCB, may also store a data set base D there.

Moreover, the default UCB contains pointers to procedures of the system formulary. This formulary, like all other formularies, contains INTNAME, VIRTUAL, CONTROL, SCRAMBLE, and UNSCRAMBLE procedures. For the system formulary, they act as follows:

**INTNAME**

INTNAME takes a natural language data description (in this case, a natural language formulary description) from TALK and transforms it into the internal name of a formulary. This is possible since formulary descriptions are stored as data in the data base.

Example: Calling INTNAME("MEDICAL";ANS) might store "1.14", the internal name for the medical formulary, in ANS.

**CONTROL**

CONTROL takes the internal name representing the formulary and decides whether user U at terminal T (U and T are in the UCB) is allowed read access to the formulary represented by the internal name.

Example: CONTROL("1.14","123") (1,"")

**VIRTUAL**

VIRTUAL takes the internal name representing the formulary and returns the virtual address
of the first word of the (five-word) formulary

SCRAMBLE No operation.

UNSCRAMBLE No operation.

TALK attaches a user to a formulary by first conversing with him to obtain a formulary name and then trying to read that formulary into the UCB. More specifically, TALK issues a call to ACCESS with parameters \(<\text{formulary name}, \text{location of UCB}\>\), and READ. Since TALK always initially attaches the user to the default UCB, the access control procedures ACCESS invokes at this time belong to the system formulary. If this READ attempt is successful, the procedure pointers of the formulary specified by the user are read into the user's UCB, destroying the system formulary pointers previously there.

The READ attempt may in fact not be successful. For example, the user may not be allowed to use the terminal, or the formulary, or the specific data set. The ACCESS program detects these violations (by using the CONTROL procedure of the default UCB) and takes appropriate actions.
4.4.1. Independence of Addressing and Access Control

After the attachment process, the User Control Block (UCB) contains the user identification U, terminal identification T, pointers to the ENTRANCE, VIRTUAL, CONTROL, SCRAMBLE, and UNSCRAMBLE procedures of a formulary, and (possibly) a data set name. Whether the user can perform read operations, write operations, etc., on a given datum is controlled by the CONTROL program. The addressing of each datum is controlled by the VIRTUAL program. Addressing of datums is now completely independent of the access control for these datums.

4.4.2. Breaking an Attachment

An existing attachment is broken (i.e., the core memory space used by the UCB filled in at that time is cleared and made available for another use) whenever

(1) the user indicates that he is finished using the information storage and retrieval system (either by explicitly declaring no or implicitly by logging out, removing a physical terminal key, reaching the end-of-job indicator in his input card deck, etc.), or

(2) the user is attached to a new formulary.
4.5. An Example of Information Storage and Retrieval Using TALK

After TALK attaches the user to the formulary he desires, information storage and retrieval operations can be carried out, as illustrated in Figure 5.
WHAT PROGRAM DO YOU WANT TO USE?

**MODALITY**

- **SPORTS PROGRAM**
- **CONTEST--PIMFAO**

WHAT PROGRAM DO YOU WANT TO USE?

**MODALITY**

- **SPORTS PROGRAM**
- **CONTEST--PIMFAO**

WHAT PROGRAM DO YOU WANT TO USE?
WHAT PROGRAM DO YOU WISH TO RUN?

EXECUTION...

MEDICAL TALK PROGRAM...

WHAT FORMULARY DO YOU WISH TO USE?

HEALTH

*** SYSTEM FORMULARY --CONTROL-- PROGRAM ***

TO USE THE HEALTH FORMULARY, YOU MUST BE PERMITTED ACCESS BY
THE COWELL STUDENT HEALTH CENTER. WHAT IS YOUR NAME?

LANCE HOFFMAN

WHAT IS YOUR SOCIAL SECURITY NUMBER?

799-97-9997

THANK YOU. YOU ARE PERMITTED TO USE THE HEALTH FORMULARY.

THE HEALTH FORMULARY HAS BEEN ATTACHED.

*** END OF SYSTEM FORMULARY --CONTROL-- PROGRAM ***

DO YOU WISH TO STORE OR RETRIEVE INFORMATION?

STORE

ANY CORRECTIONS TO EXISTING DATA URGENT

CHANGE RECORD 2549, FIELD 4 FROM 14 TO 15#

CHANGE RECORD 2586, FIELD 2 FROM F TO M#

RECORD 2549. FIELD 4 WAS 14, NOW 15.

RECORD 2586. FIELD 2 WAS M. YOU SAID IT WAS F -- REQUEST DENIED!

CHANGE RECORD 2587, FIELD 2 FROM F TO M#

RECORD 2587. FIELD 2 WAS F, NOW M.

RECORDS CHANGED: 2549, 2587.

PLEASE ADD NEW DATA.

RECORD 2612:

999-99-472, a, 12/10/47,27, wilbur arroyo, 99, 65, 327, 87#

RECORD 2613:

999-99-901, f, 2/10/47, 22, roble, psych, 26, 338, 07, 18, 47#

RECORD 2805:

999-99-872, a, 8/23/48, 26, crothers, 9, 200, 89, 53, 53, 65, 84#

***** ERROR -- FIELD 6 (DEPARTMENT) MUST BE ALPHABETIC *****

RECORD 2805:

999-99-972, a, 8/23/48, 26, crothers, law, 9, 200, 89, 53, 53, 65, 84#

RECORD 2806:


### SUMMARY OF RECORDS UPDATED OR ADDED TO DATA BASE

**5. MAR. 1969**

**UPDATED:**
- 2549 999-99-304 M 05/05/45 15 PALO ALTO
- 2587 999-99-001 F 04/14/43 20 HULNE

**MUSIC**
- 33 296 14

**POLYS**
- 14 256 207

**ADDED:**
- 2632 999-99-472 M 12/30/47 27 WILBUR ABOY
- 2633 999-99-301 F 02/26/49 22 ROBLE

**GS**
- 65 327 87

**PSYCH**
- 26 338 87

**LAW**
- 04 200 85

**ANY CORRECTIONS?**

... END MEDICAL TALK ...

WHAT PROGRAM DO YOU WISH TO RUN?

**Figure 5(b).** Conversational Storage of Data

**Figure 5.** Storage and retrieval of data in a specific system.
In this paper, we have described the formulary method of access control. This method allows the functions of data addressing and access control to be separated and performed at data access time by independent programs of arbitrary complexity. It thus represents a step forward from previous methods which used precomputed static information to determine access privileges. Using the access control procedures of a formulary, the decision on access privileges can now take account of such factors as time of day, terminal identification, and the value of the data requested (or of other data). These procedures can interact with the user to further validate his identification and his rights to access the information in question.
The author is most grateful to Professor William F. Miller for his encouragement and advice. Many other members of the Stanford Computer Science Department and the Stanford Linear Accelerator Center have also contributed their ideas and help, in particular John V. Levy and Victor Lesser. The formulary idea was initially suggested by the use of syntax definitions ("field formulations") for input/output data descriptions, as described in [Curtin53]. The library staff at the Stanford Linear Accelerator Center has been most helpful in tracking down articles on the topic. Editing of this paper was facilitated by using a text editor for the IBM 360/31 written by J. F. George.
Appendix A. Algorithms for System Programs

INTEGER PROCEDURE access(field, value, length, opn);
BEGIN COMMENT Access takes as input the natural language field name field
and does the following:
IF opn = "READ" or "READANDBLOCK", value is set to the value of field.
IF opn = "WRITE" or "WRITEANDLOCK", the value of field becomes value.
IF opn = "READANDBLOCK" or "WRITEANDLOCK" the datum is locked after the
READ or WRITE operation is performed.
VALUE has length length.
Access returns the following integers as functional values:

1 normal exit, no error
2 natural language field name/internal name correspondence cannot be
made using INTFIELD from OCB
3 opn permitted but gave error when attempted
11 opn not permitted by the CONTROL program associated with datum
 specified by field.

Note that by the time the user has left the ACCESS routine, the data may
have been updated.

IF intname(field, intnamep) = 2 THEN BEGIN j:=2; GO TO FIN END;
COMMENT return 2 if intname cannot make the correspondence;
IF opn="UNLOCKREAD" or opn="UNLOCKWRITE" THEN
BEGIN apply(opn, intnamep); COMMENT clear a read lock or a write lock;
j:=1; GO TO FIN; COMMENT and then exit;
END;
STEP:="control(intnamep,
    if opn="READ" then "GET" else
    if opn="WRITE" then "PUT" else
    if opn="READANDBLOCK" then "GETANDLOCK" else
    if opn="WRITEANDLOCK" then "PUTANDLOCK" else opn);
IF car(step)=2 THEN BEGIN COMMENT control does not allow operation;
j:=11; GO TO FIN;
END;

virtual(intnamep,datum, cdr(step));
COMMENT datum now holds the virtual address of the datum specified;
i:=try(opn,car(datum),result,length);
IF i > 2 THEN BEGIN COMMENT exit since op operation failed;
j:=1; GO TO FIN;
END;
COMMENT perform physical read (write) of size length to (from) block starting
at result;

COMMENT This next may happen after the (un)scrambled data has gone over
the line from (to) the console's computing device;
value:=apply(IF opn="READ" or opn="READANDLOCK" then scramble ELSE scramble,
result, length);
IF car(value)=1 THEN BEGIN VALUE:=cdr(value); j:=1;
COMMENT Successful completion of
Access procedure:

ELSE
  COMMENT error in performing opn so return 3; j := 3;
ENDIF:
RETURN j; COMMENT completion code;
END access:
RETURN encode

COMMENTS LOST ON TEMPORARY IF encode = 2:
IF encode = 2 THEN go TO APPEND;
else encode = f'p'1, 'p'2, 'p'3, 'p'4

ATTEMPT 1:
INTCODE encode

The value returned by an abnormal exit:

capture 1: normal exit
      p'1, p'2, p'3

capture 2: a function name

BEGIN COMMIT try 1, a source code needed to call and wait for completion of other subroutines which may need to access data that

BEGIN COMMIT try 2, a source code needed to call and wait

BEGIN SOURCE PROCEDURE f'CELL' p'1, p'2, p'3, 'p'4

BEGIN SOURCE PROCEDURE f'02' p'1, p'2, p'3, 'p'4

Page 41
Appendix B. Primitive Operations

Primitive operations are those which cannot be expressed in machine-independent form, but rather depend on the specific system and machine used. They are defined in terms of their inputs and outputs.

GET(addr, value, length)

GET returns in length storage elements starting at value the value contained in the virtual address addr. This value may be scrambled. GET returns as its own value:

1 if normal exit
2 if addr is currently blocked
4 if GET cannot access length storage elements
starting at value

PUT(addr, value, length)

PUT stores the contents of value into length storage elements starting at virtual address addr. The information stored may be scrambled. PUT returns as its own value:

1 if normal exit
2 if addr is currently blocked
4 if PUT cannot write length storage elements starting at addr

GETALOCK(image, value)

GETALOCK operates in the same way as GET except that it takes as the specification of the datum its internal name rather than a virtual address. By doing this, GETALOCK is able to

(1) simultaneously lock out internal name image to all further requests by storing it and the user identification 0 in the LOCKSTACK. It is the responsibility of the user to, when he is ready, free the locked datum (specified by image) by calling

UNLOCK(image). If this has not been done by the time the user logs off the system, it is automatically done by the system.

If user 1 locks out datum 0 which he knows by the natural language description $M_1$, and user 2 then attempts to access datum 0 (which user 2 knows by the natural language description $M_2$), the request of user 2 is denied (i.e., GET returns 2 when invoked for Image 0).
GETANDBLOCK returns as its own value:
1 if normal exit
2 if instance is currently blocked
3 if GETANDBLOCK cannot access length storage elements
starting at value

PUTANDBLOCK (iname, value)

PUTANDBLOCK is analogous to GETANDBLOCK.

UNLOCKREAD (iname)

UNLOCKREAD unlocks the read-lock which had been put on the
datum specified by the internal name iname by a previous call to
GETANDBLOCK or PUTANDBLOCK. This is done by removing the
appropriate dosblot from the LOCKSTACK. UNLOCKREAD returns as its
own value:
1 normal return
2 iname did not specify a datum locked by this user

UNLOCKWRITE (iname)

UNLOCKWRITE is analogous to UNLOCKREAD.

APPLY (x, y)

APPLY applies the function named in x when given the list y
of arguments. It returns the value of the function.

CAR (x)
CAR returns the first element of the list x.
References


