SLAC-118
UC-32
(MISC)

THE ENGINEERING OF ACCESS CONTROL MECHANISMS IN PHYSICS DATA BASES

LANCE J. HOFFMAN
STANFORD LINEAR ACCELERATOR CENTER
STANFORD UNIVERSITY
Stanford, California

PREPARED FOR THE U. S. ATOMIC ENERGY
COMMISSION UNDER CONTRACT NO. AT(04-3)-515

July 1970

ABSTRACT

A model is presented for engineering the user interface for large data base systems in order to maintain flexible access controls over experimental and other physics data. Several examples of its use are given. The model is independent of both machine and data base structure. Access control is based on sets of procedures called formularies. The decision on whether a user can read, write, update, etc., data is controlled by programs (not merely bits or tables of data) which can be completely independent of the contents or location of raw data in the data base.

The decision to grant or deny access can be made in real time at data access time, not only at file creation time as has usually 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 can be controlled at arbitrarily low levels, including the data field level and 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.

Access controls for physics data banks of the type presented herein are of special interest to regional computing centers or other centers with many independent users.

ACKNOWLEDGEMENTS

The author is deeply indebted to Professor William F. Miller for his encouragement and advice during the research and writing of the dissertation upon which this report is based. The research environment he has provided at the Stanford Linear Accelerator Center (SLAC) Computation Group makes it a pleasure to work there. His advice has been, at the same time, timely, competent, and unobtrusive.

Many other members of the Stanford Linear Accelerator Center and the Stanford Computer Science Department have also contributed their ideas and help, in particular, John Levy, Robert Tussell, and Victor Lesser. I wish to thank Professors Harold Stone, Edward Feigenbaum, and Jerome Feldman for their constructive readings of the thesis. The formulary idea was initially suggested by the use of syntax definitions ("field formularies") for input/output data descriptions, as described in Castleman [1967].

Part of the excellent research environment at SLAC is due to the very helpful and competent technical staff. My thanks go to the SLAC library for tracking down articles on the topics involved and to the SLAC Technical Information Department for translating chicken-scratches into meaningful illustrations. I appreciate the interest and assistance of Jorge Bruguera at the Stanford Computer Science Department Library. I wish to thank Kathleen Maddern for her many retypings of this report, Linda Lorenzetti, SLAC Program Librarian, and Carla West, the Executive Secretary of the SLAC Computation Group.

The dissertation upon which this report is based, "The Formulary Model for Access Control and Privacy in Computer Systems," is available as Report No. SLAC-117.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A model is presented for engineering the user interface for large data base systems in order to maintain flexible access controls over experimental and other data. Several examples of its use are given. The model is independent of both machine and data base structure. Access control is based on sets of procedures called formularies. The decision on whether a user can read, write, update, etc., data is controlled by programs (not metely bits or tables of data) which can be completely independent of the contents or location of raw data in the data base.

This type of access control is especially important in large shared systems such as regional centers or national data banks.

The decision to grant or deny access can be made in real time at data access time, not only at file creation time as has usually 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 can be controlled at arbitrarily low levels, including the data field level and 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.

Specific interpretations of the model can be implemented on any general-purpose computer; no special time-sharing or other hardware is required. The only proviso is that all requests to access the data base must be guaranteed to pass through the data base control system.

CHAPTER II

ACCESS CONTROL METHODS

A. Access Control in Existing Systems

In most current information storage and retrieval systems, access to data is controlled at the file level only — it has been tacitly assumed that all data within a file has the same value and will be altered by the same user (in particular, the file owner). This constraint is unnecessary and, in fact, inhibits optimum use of large data banks for experimental physics research. Experimental data is constantly coming into common data pools; different members of the research team use and alter this data in different ways. For example, a scanner may be able to only add new data to the data base; his supervisor may be able to alter that data; and only the experimenter may be permitted to purge that data from the data base. Few current systems allow different levels of access to fields within files. The limitations of current systems are discussed more fully in Hoffman [1970].

B. Significance of This Work to Physics Data Banks

It seemed desirable to devise a method of access control with the following characteristics:

1. It does not impose an arbitrary constraint (such as segmentation or access levels) on data or programs.
2. The method should allow efficient control of individual data elements (rather than of files or records only). Also, it should not extract unwarranted costs in storage or elsewhere from the user who wants only a small portion of his data controlled in this way.
3. The method should be independent of both machine and file structure, yet flexible enough to allow a particular implementation of it to be efficient.

Such a method would easily allow operations on physics data banks which we have not been able to carry out previously.

Examples (Note that the formulary model allows all of the examples below to be carried out simultaneously on the same data bank):

a. Scanners could, under suitable and arbitrarily complex controls, make corrections to raw data items via on-line remote terminals.

b. Massive updates or addenda of data for an experiment could be limited to the experimenter or his delegate.

c. Histories of changes in datum values can be easily kept on a log tape or other medium since all data accesses go through one central interface.

d. Historical data for experiments ("grandfather files", etc.) can only be purged by authorized personnel.

e. Using the LOCKLIST mechanism of the model, we can avoid collections of mixed data (i.e., some data items updated and others out-of-date at a given point in time).

f. Data can be manipulated only at given times, for example, only when the computer system is not heavily taxed accepting real-time data from on-line external devices.

We now present a method with all of the above characteristics.

CHAPTER III

THE FORMULARY METHOD OF ACCESS CONTROL

We now describe the "formulary" method of access control. Its salient features have been mentioned in Chapter I. The decision to grant or deny access is made at data access time, rather than at file creation 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. 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, subsumed by the formulary method. Access to individual related data items which may have logical addresses very close to each other can be controlled individually. For example, the date of an event might be unalterable while the measurement, in the adjacent memory location, is alterable.

For any particular interpretation, the installation must supply the procedures listed in Fig. 1. These procedures can all be considered a part of the general

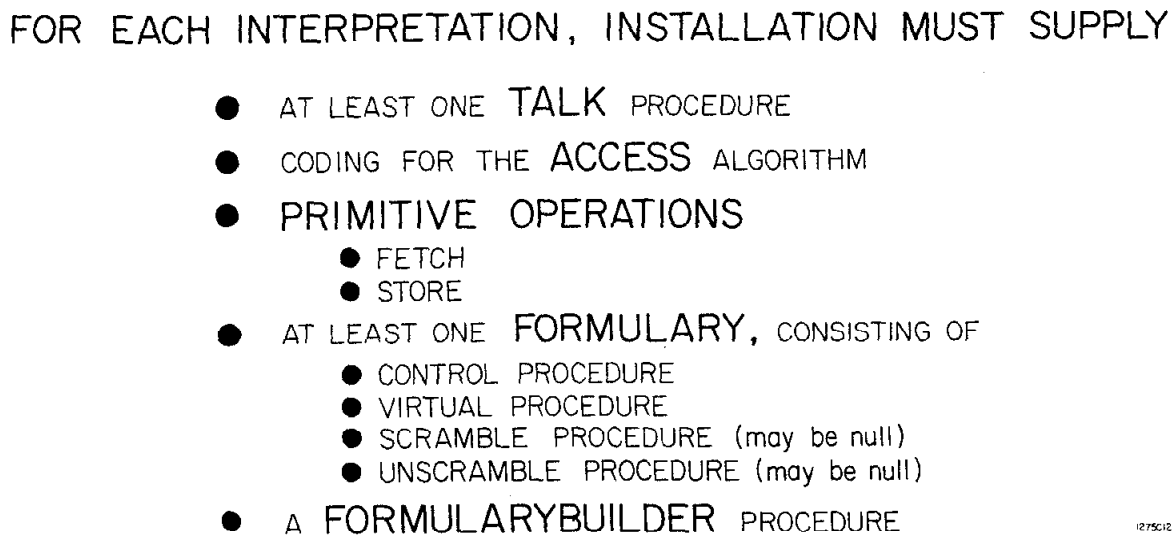


FIG. 1--Procedures supplied by the installation.

accessing mechanism, each performing a specific function. By clearly delimiting these functions, a degree of modularity is gained which enables the installation to experiment with various access control methods to arrive at the modules which

best suit its needs for efficiency, economy, flexibility, etc. This modularity also results in access control becoming independent of the remainder of the operating system, a desirable but elusive goal (Weissman [1969]). While the formulary model and its central ACCESS procedure remain unchanged, each installation can supply and easily change the procedures of Fig. 1 as desirable. They are all specified in the body of this paper. In most unclassified physics applications of the model (which is itself useful for all data bases — physics and nonphysics alike), the SCRAMBLE and UNSCRAMBLE procedures will be unnecessary and, therefore, will be null procedures.

The basic idea behind the formulary method is that a user, a terminal, and a previously built formulary (defined below) must be linked together, or attached, in order for a user to perform information storage, retrieval, and/or manipulative 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, and formulary is allowed. The general linking process is described in Section G of this chapter.

Virtual memory mapping hardware is not required to implement the model, but the model does handle systems equipped with such hardware. 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/or by the FETCH and STORE primitive operations (see Section L of this chapter) for a particular implementation.

A. Definitions and Notation

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

Examples:

1. A "tree name" such as 5.7.3.2 which denotes field 2 of branch 3 of branch 7 of branch 5 in the data base.
2. "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 Rovner and Feldman [1968].

A User Control Block, or UCB, is space in primary (core) storage allocated during the attachment process (described in Section G). It contains the user identification, terminal identification, and information about the VIRTUAL, CONTROL, SCRAMBLE, and UNSCRAMBLE procedures of the formulary the user is linked to.

Usually this information is just the virtual address of each of these procedures. 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 an appropriate action (e.g., a page fault on some computers) to move 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 described below which controls operations on any datums, including formularies. Each UCB always is in high-speed primary storage in the data area of the ACCESS procedure.

B. The ACCESS Procedure

All control mechanisms in the formulary model are invoked by a central ACCESS procedure. This ACCESS procedure is the only procedure which directly calls the primitive FETCH and STORE operations and which performs locking and unlocking operations on data items in the data base. All requests for operations on the data base must go through the ACCESS procedure.

The ACCESS procedure is a very important element of the formulary model. It never changes (see Fig. 2). It is described in full detail in Section K, and its algorithm is supplied there.

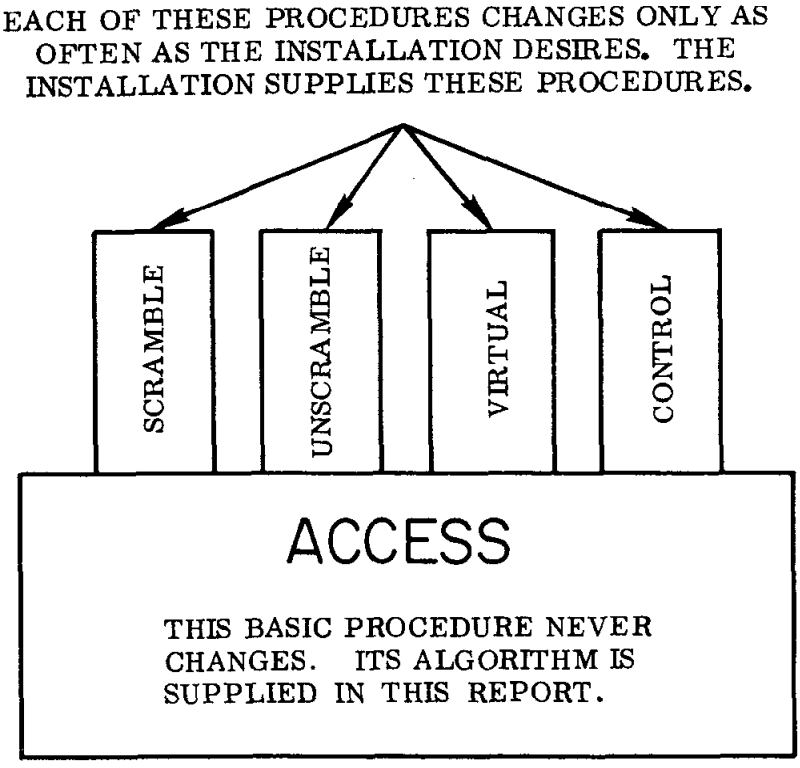


FIG. 2--Modularity of the formulary model.

The user communicates only indirectly with ACCESS. The bridge (see Fig. 3) between the system-oriented ACCESS procedure and the application-oriented user is provided by the (batch or conversational) storage and retrieval program, TALK.

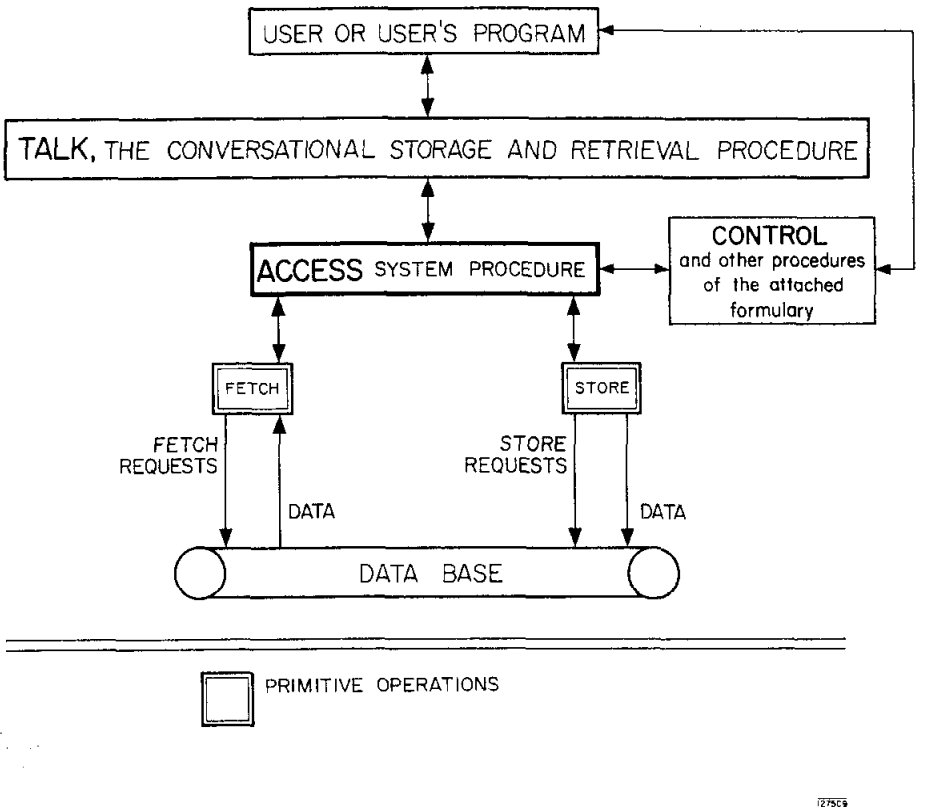


FIG. 3--User/data base interface.

C. TALK, The Application-Oriented Storage and Retrieval Procedure

To access a datum, the user must call upon TALK, the (nonsystem) application-oriented storage and retrieval procedure. TALK converses with the user (or the user's program) to obtain, along with other information, (1) a datum description in a user-oriented language, and (2) the operation the user wishes to perform on that datum (e.g., update, modify, delete, etc.). TALK translates the datum description in the user-oriented language into an internal name, thus providing a bridge between the user's conception of the data base and the system's conception of the data base. The TALK procedure is described in more detail in Section J.

D. Formularies — What They Are

A formulary is a set of procedures which controls access to information in a data base. These procedures are invoked whenever access to data is requested. They perform various functions in the storage, retrieval, and manipulation of information. The set of procedures and their associated functions are the essential elements of the formulary model of access control.

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.

1. Procedures of a Formulary

In this subsection, 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, prespecified at file makeup 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 creation time
3) more efficient use of storage.

Each procedure of a formulary should, if possible, run from memory which is alterable only under administrative control. The integrity of the system depends on the integrity of the formularies and therefore the procedures of all formularies should be written by "system" programmers who are assumed faultless. Undebugged procedures in a formulary may result in undesired types of access to data items.

A formulary has four procedures: VIRTUAL, SCRAMBLE, UNSCRAMBLE, and CONTROL. The first three are relevant but not central to access control; the decision on whether to grant the type of access desired is made solely by the

CONTROL procedure. As pointed out in Hoffman [1970], the first three procedures have been explicitly included in each formulary for three reasons:

1) to centralize in one place all functions dealing with addressing and access control;
2) to give the model the generally necessary to model existing and proposed systems; and
3) to provide well-delimited modules for cost/effectiveness studies and for experimentation with different addressing schemes and access control schemes.

a. The VIRTUAL procedure. VIRTUAL translates an internal name into the virtual address of the corresponding datum. VIRTUAL is a procedure with two input parameters:

1) the internal name to be translated
2) a cell which will sometimes be used to hold "other information" as described in Section D1d below.

VIRTUAL returns

1) the resulting virtual address
2) a completion code (1 if normal completion)

Recall that enough virtual addressing capacity is assumed 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/or by the FETCH and STORE primitive operations for a particular implementation.

b. The SCRAMBLE procedure. SCRAMBLE is a procedure which transforms raw data into encrypted form. (In most unclassified physics applications, SCRAMBLE will 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 has three output parameters:

1) a completion code (1 if normal completion)
2) the virtual address of the scrambled datum
3) the length of the scrambled datum

Note that if an auto-key cipher (one which must access the start of the ciphertext, whether or not the information desired is at the start) is used, all of the information encrypted using that cipher, be it as small as a single field or as large as an entire "file", must be governed by the same access control privileges.

Therefore, some applications may choose to use several (or many) auto-key ciphers within the same "file." It is inefficient and usually undesirable to scramble data items at other than the internal name level, e.g., scrambling as a block (to effectively increase key length) the data represented by several internal names. In cases where internal names represent data which fits into very small areas of storage, greater security may be obtained by other methods (e.g., use of nulls).

We do not discuss encrypting schemes in this paper. The interested reader is referred to Shannon [1949], Kahn [1967], and Skatrud [1969].

c. The UNSCRAMBLE procedure. UNSCRAMBLE is an unscrambling procedure which transforms encrypted data into raw form. (In most unclassified physics systems, UNSCRAMBLE will 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 has three output parameters:

1) a completion code (1 if normal completion)
2) the virtual address of the unscrambled datum
3) the length of the unscrambled datum

d. The CONTROL procedure. CONTROL is a procedure which decides whether a user is allowed to perform the operation he requests (FETCH, STORE, FETCHLOCK, etc.) on the particular datum he has specified. CONTROL may consider the identification of the user and/or the source of the request (e.g., the terminal identification) in order to arrive at a decision. CONTROL may also converse with the requesting user before making the decision.

CONTROL has two input parameters and two output parameters. The two input parameters are:

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

The two output parameters are:

1) 1 if access is allowed; otherwise an integer greater than 1
2) "other information" (explained below)

In some specific systems, data elements may themselves contain access control information. Consider three examples:

Example 1.

DATUM	R	W	30 bits of actual data

If bit R is on, DATUM is readable.

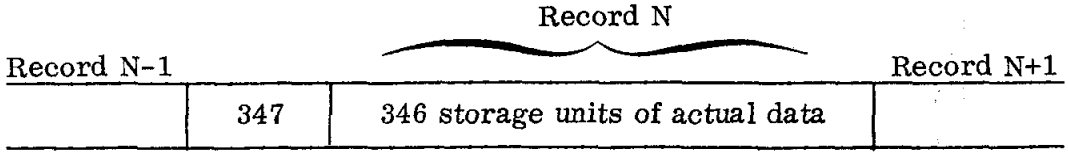
If bit W is on, DATUM is writeable.

Example 2.

ENERGY	65 BeV

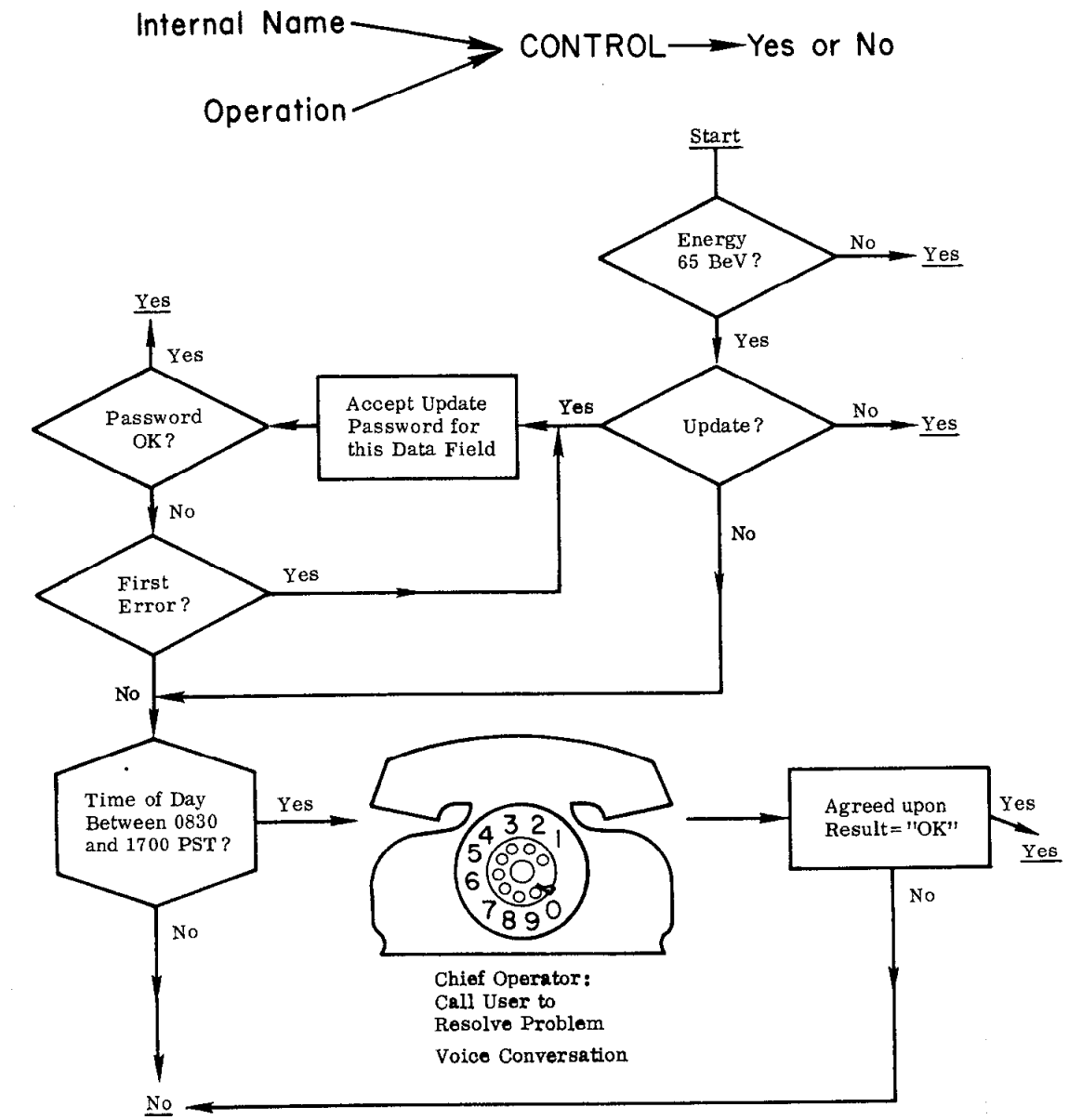
Reading or writing of energies of 65 BeV or over requires special checking. CONTROL must inspect the ENERGY cell before it can do further capability checking and eventually return 1 or some greater integer as its first output parameter (see Fig. 4). In the formulary model, CONTROL can only make a yes or no decision about access to a particular datum. Any more complex decisions, such as one involving release of a count which is possibly low enough to allow unwanted identification of individual data (e.g., "Tell me how many people in the Computation Group were tested by Health Physics for radiation symptoms last year"), can only be made by a suitably sophisticated TALK procedure. More on pitfalls involved in using counts while protecting sensitive data is given in Miller and Hoffman [1969].

Example 3.



The record contains its own length (and, therefore, also points to its successor). This type of record would appear, for example, in variable length sequential records on magnetic tape and in some list-processing applications.

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, CONTROL can (when appropriate) return the datum's virtual address as "other information." VIRTUAL, which is called after CONTROL (see the ACCESS algorithm in Section K), can then examine this "other information." If a virtual address has been put there by CONTROL,



NOTE: 1. TIME-DEPENDENT
2. FEEDBACK LOOPS
3. ON-LINE HUMAN DECISION FOR DIFFICULT PROBLEMS

1631A1

FIG. 4--A sample CONTROL procedure.

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.

Note that CONTROL can be as sophisticated a procedure as desired; it need not be merely a table-searching algorithm. Because of this, CONTROL can consider many heretofore ignored factors, in making its decision (see Fig. 4). For example, it can make decisions which are data-dependent and time-dependent. It can require two keys (or N keys) to open a lock. Also it can carry on a lengthy dialogue with the user before allowing (or denying) the access requested.

CONTROL is not limited to use at data request time. In addition to being used to monitor the interactive storage, retrieval, and manipulation of data, it can also be used at initial data base makeup time for data edit picture format checking, data value validity checking, etc. Or, alternatively, one could have two procedures CONTROL1 and CONTROL2, in two different formularies, F1 and F2. F1 could be attached at data input time and F2 at on-line storage, retrieval, manipulation, and modification time.

E. Simultaneous Use of One Formulary by Multiple Users

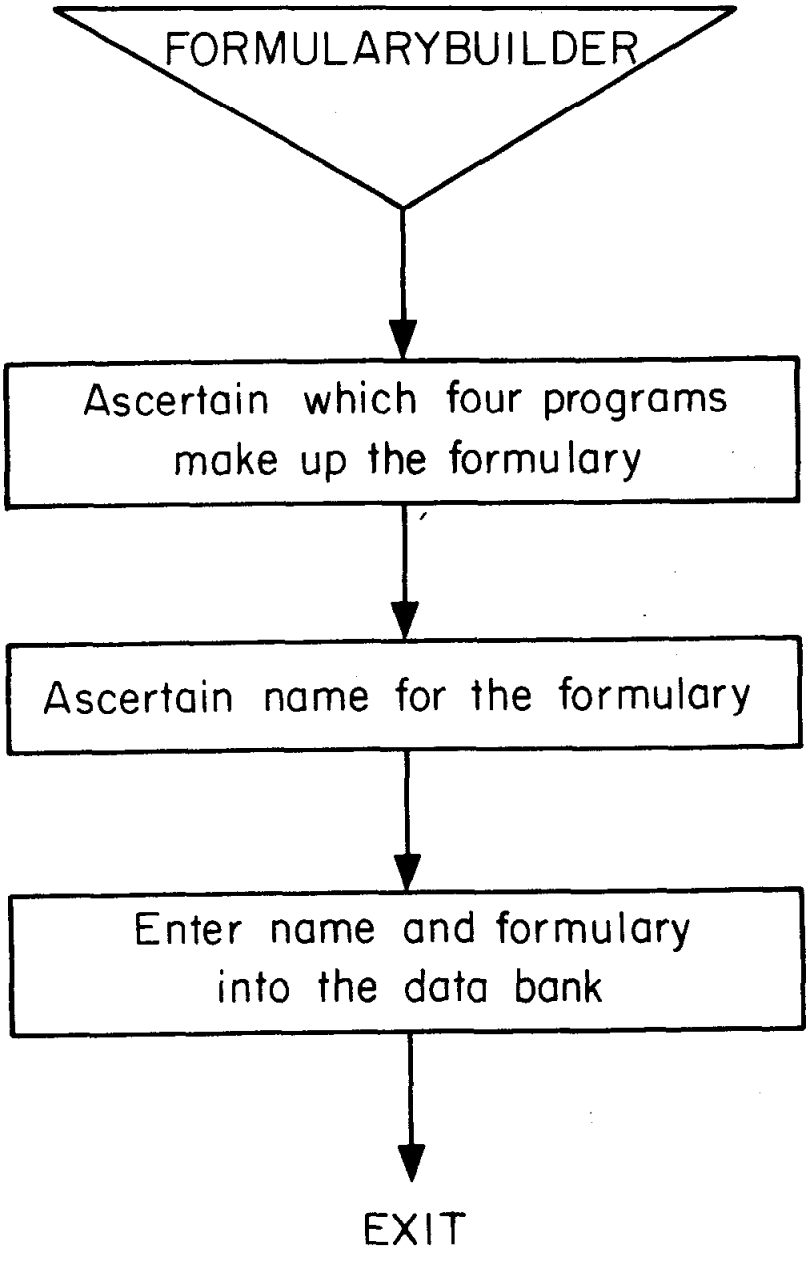
Note that the same formulary can be used simultaneously by several different users with different access permissions. This is possible because access control is determined by the CONTROL procedure of the attached formulary. This procedure can grant different privileges to different users.

F. Building a Formulary

Before a formulary can be attached to a user and a terminal, the procedures it contains must be specified. This is done using the system program FORMULARYBUILDER. FORMULARYBUILDER converses with the systems programmer 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 (see Fig. 5). The specifics of FORMULARYBUILDER depend on the particular system.

G. 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 (see Fig. 6). This linking process is done in the following manner.



1275A4

FIG. 5--General block diagram of FORMULARYBUILDER program.

THIS ATTACHMENT MUST BE MADE BEFORE ANY INFORMATION STORAGE, RETRIEVAL, OR MANIPULATION IS ALLOWED.

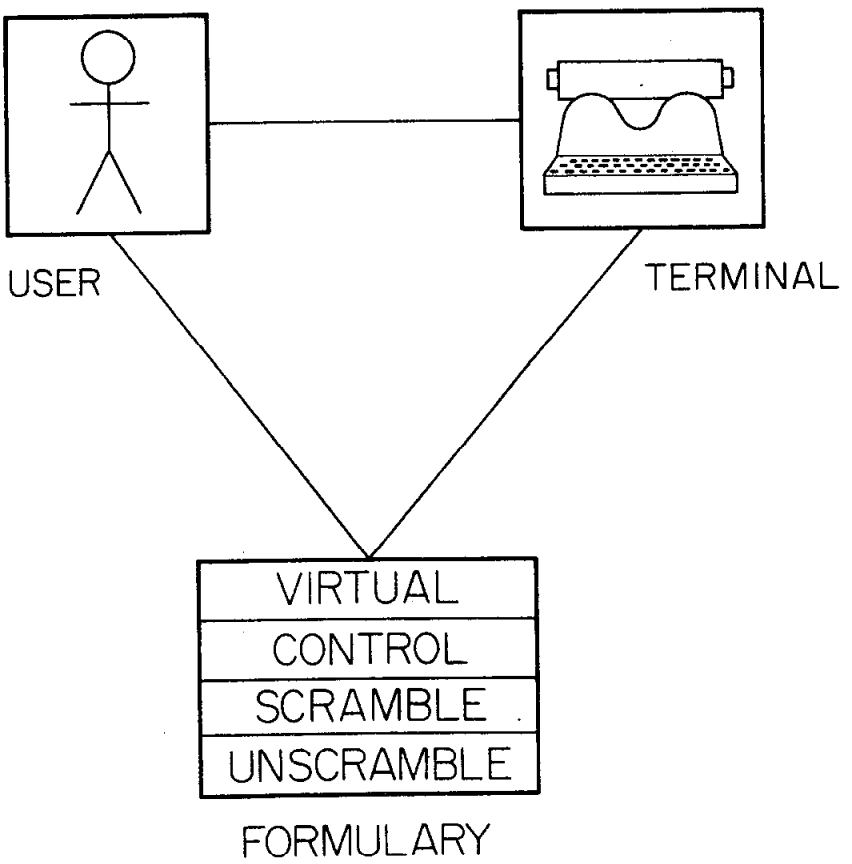


FIG. 6--Attachment of user, terminal, and formulary.

At the first time ACCESS is called (by TALK) for a given user and terminal, it will only permit attachment of a formulary to the user and terminal (i.e., it will not honor a request to fetch, store, etc.). The attachment is permitted only if the CONTROL program of the default formulary allows. The default formulary, like all other formularies, contains VIRTUAL, CONTROL, SCRAMBLE, and UNSCRAMBLE procedures. For the default formulary, they act as follows:

CONTROL	CONTROL takes the internal name representing the formulary and decides whether user U at terminal T is allowed to attach the formulary represented by the internal name. U and T are maintained in the UCB and passed to CONTROL by ACCESS.
VIRTUAL	VIRTUAL takes the internal name representing the formulary and returns the virtual address of the formulary.
SCRAMBLE	No operation.
UNSCRAMBLE	No operation.

The ATTACH attempt, if successful, causes information about the formulary specified by the user to be read into the UCB (which is located in the data area of the ACCESS procedure). ACCESS then uses this information (when it is subsequently called on behalf of this user/terminal combination) to determine which CONTROL, VIRTUAL, SCRAMBLE, and UNSCRAMBLE procedures to invoke.

1. Independence of Addressing and Access Control

After the attachment process, the User Control Block (UCB) contains the user identification U, terminal identification T, and information about (usually pointers to) the VIRTUAL, CONTROL, SCRAMBLE, and UNSCRAMBLE procedures of a formulary. Whether the user can perform certain operations on a given datum is controlled by the CONTROL program. The addressing of each datum is controlled by the VIRTUAL program. Addressing of data items is now completely independent of the access control for the data items.

2. Breaking an Attachment

An existing attachment is broken whenever

1) the user indicates that he is finished using the information storage and retrieval system (either by explicitly declaring so 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, via his TALK program, explicitly detaches himself from a formulary.

H. Subdivision of Data Base into Files Not Required

Note that while the concept of a data set (or a "file") MAY be used, the formulary method does not require this. This represents a significant departure from previous large-scale data base systems which were nearly all organized with files (data sets) as their major subdivisions. Under the formulary scheme, access to information in a data set is not governed by the data set name. Rather, it is governed by the CONTROL procedure of the attached formulary. Similarly, addressing of data in a data set is governed by the VIRTUAL procedure and not by the data set name. Subdividing a data base into data sets, while certainly permitted and often desirable, is not required by the formulary model.

I. Concurrent Requests to Access Data — The LOCKLIST

The problem of two or more concurrent requests for exclusive data access necessitates a mechanism to control these conflicts among competing users. This problem has been discussed, and solutions proposed, in Dijkstra [1965], Hsiao [1968], and Shoshani and Bernstein [1969]. In the formulary model, data can be set aside (locked) dynamically for the sole use of one user/terminal combination in a manner similar to Hsiao's "blocking" (Hsiao [1968]), using a mechanism known as the LOCKLIST.

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 access rights considerations only. Locking and unlocking are handled by a separate mechanism, the LOCKLIST. The LOCKLIST is a list of triplets maintained by the ACCESS program and manipulated by the FETCHLOCK, STORELOCK, UNLOCKFETCH, and UNLOCKSTORE operations. Each triplet contains (1) the internal name of a current item, (2) the identification of the user/terminal combination which caused it to be locked, and (3) the type of lock (fetch or store). Any datum represented by a triplet on the LOCKLIST can be accessed only by the user/terminal combination which caused it to be locked.

Data items which can be locked are atomic, i.e., subparts of these data items can not be locked. This implies, for example, that if a user wishes to lock a tree structure and then manipulate the tree without fear of some other

user changing a subnode of the tree, either

1) The tree must be atomic in the sense that its subnodes do not have internal names in the data base system, or
2) each subnode must be explicitly locked by the user and only after all of these are locked can he proceed without fear of another user changing the tree.*

J. The TALK Procedure — Details

To access a datum, the user must effectively call upon TALK, the (nonsystem) application-oriented storage and retrieval procedure. TALK converses with the interactive user and/or the user's program and/or the operating system to obtain

1) a datum description in a user-oriented language
2) the operation the user wishes to perform on that datum
3) user identification and other information about the user and/or the terminal where the user is located.

Depending on the particular system, the user explicitly gives TALK zero, one, two, or all three of the above parameters. TALK supplies the missing parameters (if any), converts (1) to an internal name, and then passes the user identification, the terminal identification, the internal name of the datum, and the desired operation to the ACCESS procedure, which actually attempts to perform the operation.

Note that one system may have available many TALK procedures. A user requests invocation of any of them in the same way he initiates any (nonsystem) program. Sophisticated users will require only "bare-bones" TALK procedures, while novices may require quite complex tutorial TALK procedures. They may both be using the same data base while availing themselves of different datum descriptions. As an example, one TALK procedure might translate English "field names" into internal names, while another TALK procedure translates French "field names" into internal names. This ability to use multiple and user-dependent descriptions of the same item is not available with such generality in any system the author is aware of, though some systems allow lesser degrees of this (Jones [1968], Giering [1967]).

---

*A more general and elegant method of handling concurrent requests to access data is being developed by R. D. Russell at SLAC as part of a general resource allocation method. Much of the housekeeping work currently done in the formulary model can be handled by his method.

The above remarks about using different TALK procedures also apply if a system uses only one relatively sophisticated TALK procedure which takes actions dependent on the person or terminal using it at a given time.

K. The ACCESS Procedure — Details

ACCESS uses the VIRTUAL, CONTROL, UNSCRAMBLE, and SCRAMBLE procedures specified in the UCB to carry out information storage and retrieval functions. Its input parameters are:

1) information about the user, terminal, etc., defined by the installation. This information is passed by the procedure that calls ACCESS;
2) internal name of datum;
3) an area which either contains or will contain the value of the datum specified by (2);
4) the length of (3);
5) operation to perform — FETCH, FETCHLOCK, STORE, STORELOCK, UNLOCKFETCH, UNLOCKSTORE, ATTACH, or DETACH. FETCHLOCK and STORELOCK lock datums to further fetch or store accesses respectively (except by the user/terminal combination for which the lock was put on). UNLOCKFETCH and UNLOCKSTORE unlock these locks. ATTACH and DETACH respectively create and destroy user/terminal/formulary attachments.
6) a variable in which a completion code is returned by ACCESS.

ACCESS itself handles all operations of (5) except FETCH and STORE. For FETCH and STORE operations on the data base, it invokes the FETCH and STORE primitives specified in Section L.

An ALGOL algorithm for the ACCESS procedure follows. This procedure is quite important and should be examined carefully. The comments in the algorithm should not be skipped, as they often suggest alternate methods for accomplishing the same goals. An equivalent FORTRAN algorithm is given as Appendix A. Note that some means must be provided to determine which formulary is attached so that the CONTROL, SCRAMBLE, UNSCRAMBLE, and VIRTUAL procedures of that particular formulary can be invoked. The program of Appendix A transfers this responsibility to those procedures themselves, which determine which formulary is attached by examining COMMON data set up previously by the ACCESS procedure. An alternative method, if ACCESS were written in a more powerful language or in assembly language, would be to use a transfer vector.

Note that two procedures and their corresponding calls can be removed from ACCESS if no user will ever have to lock out access to a datum which ordinarily can be accessed by several users at the same time or if the installation wishes to use another method to control conflicts among users competing for exclusive access to datums; this makes the procedure considerably shorter. Such a "no parallelism" version of the ACCESS algorithm is given in Appendix B.

The ACCESS Algorithm

```
procedure access (info, intname, val, length, opn, compcode);
integer array info, val; integer, length, opn, compcode;
begin comment If OPN = FETCH, VAL is set to the value of the datum
              represented by INTNAME.
              If OPN = STORE, the value of the datum represented by
              INTNAME is replaced by the value in the VAL array.
              If OPN = FETCHLOCK or STORELOCK, the datum is locked to
              subsequent FETCH or STORE operations by other users or from
              other terminals until an UNLOCKFETCH or UNLOCKSTORE operation,
              whichever is appropriate, is performed.
              If OPN = UNLOCKFETCH or UNLOCKSTORE, the fetch lock or store
              lock previously inserted by a FETCHLOCK or STORELOCK opera-
              tion is removed.
              If OPN = ATTACH, the formulary represented by internal name
              INTNAME is attached to the user and terminal described in the
              INFO array.
              If OPN = DETACH, the formulary represented by internal name
              INTNAME is detached from the user and terminal described
              in the INFO array.
              VAL is LENGTH storage elements long.
              Note that a FETCH (STORE) operation will actually attempt
              to fetch (store) LENGTH storage elements of information.
              It is the responsibility of the TALK procedure to handle
              scrambling or unscrambling algorithms that return outputs
              of a different length than their inputs.
```

ACCESS returns the following integer completion codes in COMPCODE:

1 normal exit, no error

2 unlock operation requested by user or terminal who/which did not set lock

3 operation permitted but gave error when attempted

4 attempt to unlock datum which is not locked in given manner

5 cannot handle any more User Control Blocks (would cause table overflow)

6 attempt to detach nonexistent user/terminal/formulary combination

7 operation permitted for this user and terminal but could not be carried out since datum was locked (by another user/terminal) to prevent such an operation

8 cannot put lock on as requested since LOCKLIST is full

9 datum already locked by this user and terminal

10 error return from VIRTUAL procedure

11 operation on the datum represented by INTNAME not permitted by CONTROL procedure of the attached formulary

12 end of data set encountered by FETCH operation.

Note that by the time the user has left the ACCESS routine, the data may have been changed by another user (if the original user did not lock it). Note that ACCESS could be altered to allow scrambling and unscrambling to take place at external devices rather than in the central processor.

Important: ACCESS expects the following to be available to it. The installation supplies these in some way other than as parameters to ACCESS (for example, as global variables in ALGOL or COMMON variables in FORTRAN) —

(1) ISTDUCB — the default User Control Block. Its length is NUCB storage units.

(2) NUCB — see (1).

(3) UCB — a list of User Control Blocks (UCB's) initialized outside ACCESS to ucb (1, 1) = -2,

ucb (i, j) = anything when $\sim(i = j = 1)$

UCB is declared as integer array (1:maxusers, 1:nucb).

(4) MAXUSERS — the maximum number of users which can be actively connected to the system at any point in time.

(5) ITALK — the length of the INFO array (which is the first parameter of ACCESS) — INFO contains information about the user and terminal which is used by ACCESS and also passed by ACCESS to procedures of the attached formulary. INFO(1) contains user identification.

(6) LOCKLIST — a list of locks (each element of the LOCKLIST array should be initialized outside ACCESS to -1).
LOCKLIST is declared as integer array (1:4, 1:maxllist).

(7) MAXLLIST — the maximum length of the LOCKLIST

(8) CS1 — a semaphore to govern simultaneous access to the critical section of the ACCESS procedure (initialized to 1 outside ACCESS).

ACCESS assumes that the variables FETCH, STORE, FETCHLOCK, STORELOCK, UNLOCKFETCH, UNLOCKSTORE, ATTACH, and DETACH have been initialized globally and are never changed by the installation;

integer array iucb [1:nucb], reslt [1:length];

integer i, ii, islot, j, yesno, other, n, datum;

integer procedure testandset (semaphore); integer semaphore;
begin comment TESTANDSET is an integer function designator. It returns -1 if SEMAPHORE was in the state LOCKED on entry to TESTANDSET. Otherwise, TESTANDSET returns something other than -1. In all cases, SEMAPHORE is in state LOCKED after the execution of the TESTANDSET procedure, and must be explicitly unlocked in order for it to be used again.

TESTANDSET is used to implement a controlling mechanism to prevent conflicts among users competing for the same resource, as discussed in (Dijkstra [1965]). It will not prevent "deadly embraces" (Habermann [1969]). No explicit code is given here, since the function is machine-dependent. The manner in which TESTANDSET is implemented for a particular machine, the IBM 360/67, is shown in the listing of the TESTSE procedure of Appendix A.

This procedure can be removed if no user will ever have to lock out access to a datum which ordinarily can be accessed by several users at the same time or if the installation wishes to use another method to control conflicts among users competing for exclusive access to datums;

< code >

end testandset;

integer procedure idxll (intname, opn); integer intname, opn;
begin comment IDXLL, given an internal name INTNAME, returns the relative position of INTNAME on the LOCKLIST if the datum represented by INTNAME is locked in a manner affecting the operation OPN. Otherwise, IDXLL returns

the negation of the relative location of the first empty slot on the LOCKLIST. If the LOCKLIST is full and the INTNAME/OPN combination is not found on it, IDXLL returns 0.

This procedure can be removed if no user will ever have to lock out access to a datum which ordinarily can be accessed by several users at the same time or if the installation wishes to use another method to control conflicts among users competing for exclusive access to datums;

```
integer firstempty;
j := if opn = FETCH or opn = UNLOCKFETCH or opn = FETCHLOCK then 1 else 2;
idxll := firstempty := 0;
for i := 1 step 1 until maxllist do
    begin ii := -i;
    if locklist [1, i] = -1 then firstempty := i
else if locklist [1, i] = intname and locklist [2, i] = j then begin idxll := i;
                                                        go to RET
                                                        end;
    end;
if firstempty ≠ 0 then idxll := -firstempty;
RET:
end idxll;

procedure ret (i); integer i;
begin comment RET sets the completion code compcode to i and then causes
exit from the ACCESS procedure;
compcode := i; go to FIN
end ret;
```

```
compcode := 1;
comment first let's see if we recognize the user/terminal combination
in INFO;
islot := 0;
for i := 1 step 1 until maxusers do
    begin ii := i;
        if ucb [i, 1] = -2 then begin comment end of list of ucb's;
                    if islot=0 then begin if ii ≠ maxusers then ucb [ii+1, 1]:= - 2;
                                    go to XFER;
                                    end
                                else go to PRESETUP;
                                end
else if ucb [i, 1] =-1 then islot := ii
    comment remember this slot if vacant;
else begin for j := 1 step 1 until italk do
        if ucb [i, j]≠info[j] then go to ILOOPND;
    go to SETUPPTRS
    end;
ILOOPND:
    end i loop;
if islot = 0 then ret(5); comment cannot handle any more UCBs;
PRESETUP:
ii := islot;
XFER:
for k := 1 step 1 until italk do ucb[ii, k] := info[k];
for k := italk + 1 step 1 until nucb do ucb[ii, k] := istducb[k];
```

SETUPPTRS:

for i := 1 step 1 until nucb do iucb[i] := ucb[ii, i];

comment set up pointers to appropriate user control block for particular implementation. Note well: Setting up pointers to appropriate user control blocks is quite dependent on the particular system. For an example of one implementation, see Appendix A;

comment We have now associated user and terminal with the user control block (representing a formulary) in relative position i of the UCB table;

if iucb[nucb] ≠ intname and opn = DETACH then ret (6);

comment attempt to detach user/terminal/formulary combination not currently attached;

control (intname, opn, yesno, other);

if yesno > 1 then ret (11);

comment return 11 if CONTROL does not permit operation;

if opn = ATTACH then begin ucb[ii, nucb] := intname; go to FIN

end;

comment Note well: In many implementations, pointers to each procedure of the formulary (obtained by having VIRTUAL transform intname into a virtual address) might be put into the UCB upon attachment. In others, the philosophy used here of only putting one pointer — to the formulary — into the UCB will be followed. The decision should take into account design parameters such as implementation language, storage available, etc.;

if opn = DETACH then begin comment detach formulary (this leaves an open slot in the ucb array); ucb[ii, 1] := -1; go to FIN

end;

```
if opn = UNLOCKFETCH or opn = UNLOCKSTORE then
    begin i := idxll(intname, opn); comment find internal name on LOCKLIST;
    if i ≤ 0 then ret(4); comment cannot find it;
    for j := 1 step 1 until italk do
        if locklist [2+j, i] ≠ iucb[j] then ret(2);
    locklist[1, i] := -1; comment undo the lock and mark slot in UCB array empty;
    go to FIN
    end unlock operation;
TRY:
if testandset(cs1) = -1 then go to TRY;
comment loop until no other user is executing the critical section below;
comment ACCESS should ask to be put to sleep if embedding system permits;
comment ---------- enter critical section for locking out datums --------;
i := idxll(intname, opn);
comment get relative location of locked datum in locklist;
if i > 0 then begin comment datum found on locklist so see if it was locked by
                this user and terminal;
                for j := 1 step 1 until italk do
                    if locklist [2+j, i] ≠ iucb[j] then ret(7);
                    comment data already locked by another user or terminal;
                    if opn = FETCHLOCK or opn = STORELOCK then ret(9);
                    comment datum already locked by this user and terminal,
                    so return completion code of 9;
                end;
```

```
i := -i;
if opn = FETCHLOCK or opn = STORELOCK then
    begin comment this is a lock operation;
    if i = 0 then ret(8); comment connot set lock since locklist is full;
    locklist[2, i] := if opn = FETCHLOCK then 1 else 2;
    comment set appropriate lock;
    for j := 1 step 1 until italk do locklist[2+j, i] := iucb[j];
    comment place user and terminal identification into LOCKLIST;
    locklist[1, i] := intname; comment place internal name on LOCKLIST;
    go to FIN;
    end lock operation;
virtual (intname, datum, other, compcode);
comment VIRTUAL returns in datum the virtual address of the datum specified;
if compcode > 1 then ret(10); comment error return from VIRTUAL;
if opn = STORE then
    begin comment store operation;
    scramble (val, length, compcode, reslt, n);
    if compcode > 1 then ret(3);
    comment operation permitted but gave error when attempted;
    comment now perform a physical write of n storage units to the block
    starting at reslt;
    store (datum, reslt, n, compcode);
    if compcode > 1 then ret(3)
    end
else
    begin comment fetch operation;
    fetch (datum, reslt, length, compcode);
```

```
        if compcode = 2 then ret(12); comment end of data set encountered;
        if compcode > 1 then ret(3);
        unscramble (reslt, length, compcode, val, n);
        if compcode > 1 then ret(3);
        end fetch operation;
FIN:
comment ---------- Leave critical section for locking out datums ----------;
cs1 := 1;
end access;
```

L. FETCH AND STORE Primitive Operation

The two primitive operations FETCH and STORE are supplied by the installation. These primitives actually perform the physical reads and writes which cause information transfer between the media the data base resides on and the primary storage medium (usually, magnetic core storage). They are invoked only by the ACCESS procedure.

The primitive operations cannot be expressed in machine-independent form, but rather depend on the specific system and machine used. They are defined functionally below.

FETCH(ADDR, VALUE, LENGTH, COMP)

This primitive fetches the value which is contained in the storage locations starting at virtual address ADDR and returns it in VALUE. This value may be scrambled, but if so unscrambling will be done later by UNSCRAMBLE (called from ACCESS), and LENGTH is the length of the scrambled data. The value comprises LENGTH storage elements. Upon completion, the completion code COMP is set to:

1 if normal exit
2 if end of data set encountered when physical read attempted
3 if length too big (installation-determined)
4 if illegal virtual address given to fetch from
5 if error occurred upon attempt to do physical read

STORE(ADDR, VALUE, LENGTH, COMP)

This primitive stores LENGTH storage elements starting at virtual address VALUE into LENGTH storage elements starting at virtual address ADDR. The information stored may be scrambled, but if so the scrambling has already been done by SCRAMBLE (called from ACCESS), and LENGTH is the length of the scrambled data. Upon completion, the completion code COMP is set to:

1 if normal exit
3 if length too big (installation-determined)
4 if illegal virtual address given to store into
5 if error occurred upon attempt to do physical write.

CHAPTER IV

CONCLUSIONS

A. Summary

We have defined and demonstrated a model of access control which allows real-time decisions to be made about privileges granted to users of a data base. Raw data need appear only once in the data base and arbitrarily complex access control programs can be associated with arbitrarily small fragments of this data.

The desirable characteristics for an access control method laid out in Chapter II are all present (though we have not yet run enough experiments to make general statements about efficiency):

1) No arbitrary constraint (such as segmentation or sensitivity levels) is imposed on data or programs
2) The method allows control of individual data elements. Its efficiency depends on the specific system involved and the particular controls used. As shown in Hoffman [1970], very little performance degradation due to increased overhead was added by the introduction of formularies to the tape-based system discussed there.
3) No extra storage or time is required to describe data which the user does not desire to protect.
4) The method is machine-independent and also independent of file structure. The efficiency of each implementation depends mainly on the adequacy of the formulary method for the particular data structures and application involved.

B. Future Work

More experiments should be carried out to determine the amount of additional system overhead introduced by user formularies. This will vary over data structures and over data base systems. In particular, actual costs in additional central processor cycles should be determined for various hardware systems.

Criteria of system efficiency, degree of control required, etc., should be developed to determine the extent of usefulness of the formulary method. Some preliminary work has already been done in this area (Wortman and Hoffman [1969]).

Using the formulary method, cost measures for scrambling and unscrambling techniques and for threat monitoring (Hoffman [1969]) subsystems can be developed in the same manner that some cost measures were developed in Hoffman [1970].

To observe the full capabilities of the method and its potential for storage efficiency, a system should be developed where quite a number of users share several formularies. Also, the problem of users granting limited capabilities to other users, these new users granting even more limited capabilities to still other users, etc., and all this being done while access control decisions are being made in real time by procedures, should be investigated in more detail. Once this problem of granting limited privileges is solved, we will see much more controlled sharing of mutually useful programs and data. The implications here for giant physics-oriented data banks are very great.

A most promising area for future work is the development of a generalized resource allocation system which incorporates the formulary model as a first stage and a sophisticated scheduler as a second stage. Such a system is currently being investigated by R. D. Russell at SLAC.

Finally, since the central ACCESS procedure is fixed, hardware or microprogrammed implementations of it could be built which would greatly decrease the overhead in central processor cycles involved in using the formulary method.

REFERENCES

Castleman, P. A. [1967]. "User-defined syntax in a general information storage and retrieval system," in Information Retrieval, The User's Viewpoint, An Aid to Design, International Information, Inc.

Dijkstra, E. W. [1965]. Cooperating sequential processes. Department of Mathematics, Technological University, Eindhoven, The Netherlands.

Hoffman, Lance J. [1969]. Computers and privacy: A survey. Computing Surveys 1, 2 (June 1969).

Hoffman, Lance J. [1970]. The Formulary Model for Access Control and Privacy in Computer Systems. Report No. SLAC-117, Stanford Linear Accelerator Center.

Hsiao, D. K. [1968]. A File System for a Problem Solving Facility. Ph.D. Dissertation in Electrical Engineering, Univ. of Pennsylvania, Philadelphia.

Jones, R. S. [1968]. DATA FILE TWO — A data storage and retrieval system. Proc. SJCC 1968, 171-181.

Kahn, D. [1967]. The Codebreakers. MacMillan, New York.

Miller, W. F. and Hoffman, L. J. [1969]. A method of extracting record-specific information from "statistical" data banks. CGTM-67, Stanford Linear Accelerator Center, Computation Group, Stanford, California.

Rovner, P. D. and Feldman, J. A. [1968]. The Leap Language and data structure. Proc. IFIP Congress 1968, C73-C77.

Shannon, C. E. [1949]. Communication theory of secrecy systems. Bell System Tech. J. 28, 656-715.

Shoshani, A. and Bernstein, A. J. [1969]. Synchronization in a parallel-accessed data base. Comm. ACM 12, 11 (November 1969), 604-607.

Skatrud, R. O. [1969]. The application of cryptographic techniques to data processing. Proc. AFIPS 1969 Fall Joint Computer Conference, 111-117.

Wortman, David and Hoffman, Lance J. [1969]. Steps toward a formalism for the formulary model. CGTM-83, Stanford Linear Accelerator Center, Computation Group, Stanford, California.

APPENDIX A -- FORTRAN VERSION OF ACCESS PROCEDURE

```
      SUBROUTINE ACCESS(INFO,INTNAME,VALUE,LENGTH,OPN,COMPCODE)          00524500
C                                                                        00524600
C THIS PROCEDURE TAKES AS INPUT THE INTERNAL NAME INTNAME AND            00524700
C DOES THE FOLLOWING:                                                    00524800
C                                                                        00524900
C IF IOPN=FETCHP, VALUE IS SET TO THE VALUE OF THE                       00525000
C DATUM REPRESENTED BY INTNAME.                                          00525100
C                                                                        00525200
C IF IOPN=STOREP, THE VALUE OF THE DATUM REPRESENTED                     00525300
C BY INTNAME BECOMES VALUE.                                              00525400
C                                                                        00525500
C IF IOPN=FLOCKP, SLOCKP, UNLFEP, OR UNLSTP, THE DATUM                   00525600
C REPRESENTED BY INTNAME IS RESPECTIVELY LOCKED TO FUTURE                00525700
C FETCHES, LOCKED TO FUTURE STORES, UNLOCKED TO FUTURE                   00525800
C FETCHES, OR UNLOCKED TO FUTURE STORES.                                 00525900
C (LOCKING A DATUM LOCKS OUT ALL USER/TERMINAL COMBINATIONS              00526000
C  EXCEPT THE ONE THAT SET THE LOCK.)                                    00526100
C                                                                        00526200
C THE LENGTH OF VALUE IS LENGTH.                                         00526300
C                                                                        00526400
C ACCESS RETURNS IN COMPCODE THE FOLLOWING COMPLETION CODES:             00526500
C                                                                        00526600
C           1 NORMAL EXIT, NO ERROR                                      00526700
C           2 UNLOCK OPERATION REQUESTED BY USER/TERMINAL                00526800
C             WHO/WHICH DID NOT SET LOCK                                 00526900
C           3 IOPN OPERATION PERMITTED BUT GAVE ERROR WHEN ATTEMPTED     00527000
C           4 ATTEMPT TO UNLOCK DATA WHICH IS NOT LOCKED IN GIVEN MANNER 00527100
C           5 CANNOT HANDLE ANY MORE USER CONTROL BLOCKS                 00527200
C           6 ATTEMPT TO DETACH NONEXISTENT USER/TERMINAL/FORMULARY      00527300
C             COMBINATION                                                00527400
C           7 IOPN OPERATION PERMITTED BUT WAS UNABLE TO BE CARRIED OUT  00527500
C             SINCE THE DATUM WAS LOCKED TO PREVENT SUCH AN OPERATION    00527600
C           8 CANNOT PUT ON LOCK AS REQUESTED SINCE LOCKLIST IS FULL     00527700
C           9 DATUM ALREADY LOCKED BY THIS USER AND TERMINAL             00527800
C          10 VIRTUAL PROCEDURE CANNOT TRANSLATE INTERNAL NAME INTO      00527900
C             VIRTUAL ADDRESS                                            00528000
C          11 IOPN OPERATION NOT PERMITTED ON DATUM REPRESENTED          00528100
C             BY INTNAME; DETECTION CARRIED OUT BY THE CONTROL           00528200
C             PROGRAM OF THE ATTACHED FORMULARY                          00528300
C          12 END OF DATA SET ENCOUNTERED ON FETCH ATTEMPT               00528400
C                                                                        00528500
C                                                                        00528600
C                                                                        00528700
C FORMAT OF LOCKLIST (LLIST) IS:                                         00528800
C                                                                        00528900
C    ENTRY 1   ENTRY 2    ...   ENTRY N   ENTRY N+1 ... ENTRY 100       00529000
C INAME                                                                  00529100
C OPN                                                                    00529200
C USER/TERMINAL INFORMATION                                              00529300
C                                                                        00529400
C OPERATIONS --                                                          00529500
C  1 FETCH, 2 STORE, 3 BOTH FETCH AND STORE                              00529600
C INAME=-1 IMPLIES THAT SLOT ON LOCKLIST IS EMPTY                        00529700
C                                                                        00529800
C                                                                        00529900
C                                                                        00530000
      IMPLICIT INTEGER(A-Z)                                              00530100
      COMMON/CURUCB/IUCB                                                 00530200
      COMMON/CONSTANTS/NUCB,NFORM,MAXUSERS,MAXLLIST,ITALK,               00530300
     1   FORM1,FORM2,FORM3,                                              00530400
     2   NEXTALL,SAMEALL,                                                00530500
     3   FETCHP,STOREP,UNLFEP,UNLSTP,FLOCKP,SLOCKP,ATTACHP,DETACHP       00530600
      COMMON/UCB/ISTDUCB                                                 00530700
      COMMON/OWN1/UCB1,LLIST,CS1                                         00530800
      INTEGER LLIST(4,100)                                               00530900
      INTEGER UCB1(100,3)                                                00531000
C ************* 100 IS MAXUSERS, NUCB IS 3                             00531100
      INTEGER ISTDUCB(3)                                                 00531200
      INTEGER INFO(ITALK)                                                00531300
      INTEGER VALUE(20)                                                  00531400
C **** DIMENSION IS LENGTH STORAGE ELEMENTS, IN THIS CASE 80            00531500
C STORAGE ELEMENTS.  THIS MUST BE SPECIFIED AS 20 FORTRAN ELEMENTS DUE   00531600
C TO REQUIREMENTS OF THE FORTRAN LANGUAGE.                               00531700
      INTEGER INTNAME,LENGTH,OPN,COMPCODE                                00531800
C                                                                        00531900
      INTEGER IUCB(3)                                                    00532000
C DIMENSION SHOULD BE NUCB BUT FORTRAN DOES NOT ALLOW THAT CONSTRUCTION  00532100
      INTEGER RESLT(20)                                                  00532200
C **** DIMENSION IS LENGTH STORAGE ELEMENTS, IN THIS CASE 80            00532300
C STORAGE ELEMENTS.  THIS MUST BE SPECIFIED AS 20 FORTRAN ELEMENTS DUE   00532400
C TO REQUIREMENTS OF THE FORTRAN LANGUAGE.                               00532500
C                                                                        00532600
```

The ACCESS procedure has the following characteristics:

a. only procedure which directly calls FETCH and STORE primitives.
b. only procedure which performs locking and unlocking operations.
c. all requests for operations on data base must go through it.

Lines 5247-5284 above describe the operation of the ACCESS procedure.

```
      COMPCODE=1                                                        00532700
      ISLOT=0                                                           00532800
C FIRST TRY TO RECOGNIZE USER/TERMINAL COMBINATION IN INFO ARRAY        00532900
      DO 1 I=1,MAXUSERS                                                 00533000
      II=I                                                              00533100
      IF (UCB1(I,1) .EQ. -2) GO TO 2                                    00533200
C END LIST OF UCBS                                                      00533300
      IF (UCB1(I,1) .EQ. -1) GO TO 3                                    00533400
      DO 4 J=1,ITALK                                                    00533500
      IF (UCB1(I,J) .NE. INFO(J)) GO TO 1                               00533600
    4 CONTINUE                                                          00533700
      GO TO 6                                                           00533800
    2 IF (ISLOT .NE. 0) GO TO 7                                         00533900
      IF (II .NE. MAXUSERS) UCB1(II+1,1)=-2                             00534000
      GO TO 16                                                          00534100
    3 ISLOT=II                                                          00534200
C REMEMBER THIS SLOT IF VACANT                                          00534300
    1 CONTINUE                                                          00534400
      IF (ISLOT .EQ. 0) GO TO 805                                       00534500
C CANNOT HANDLE ANY MORE UCBS                                           00534600
    7 II=ISLOT                                                          00534700
   16 DO 5 K=1,ITALK                                                    00534800
    5 UCB1(II,K)=INFO(K)                                                00534900
      K1=ITALK+1                                                        00535000
      DO 8 K=K1,NUCB                                                    00535100
    8 UCB1(II,K)=ISTDUCB(K)                                             00535200
    6 DO 9 I=1,NUCB                                                     00535300
    9 IUCB(I)=UCB1(II,I)                                                00535400
C SET UP POINTERS TO APPROPRIATE USER CONTROL BLOCK                     00535500
C USER AND TERMINAL NOW ASSOCIATED WITH POSITION II OF UCB TABLE.       00535600
      IF((IUCB(NUCB) .NE. INTNAME).AND. (OPN .EQ. DETACHP)) GO TO 806   00535700
C ATTEMPT TO DETACH USER/TERMINAL/FORMULARY COMBINATION NOT CURRENTLY    00535800
C ATTACHED                                                              00535900
      CALL CONTROL(INTNAME,OPN,YESNO,OTHER)                             00536000
      IF (YESNO .GT. 1) GO TO 811                                       00536100
C RETURN 11 IF CONTROL DOES NOT PERMIT OPERATION                        00536200
      IF (OPN .EQ. ATTACHP) GO TO 10                                    00536300
      IF (OPN .EQ. DETACHP) GO TO 11                                    00536400
      IF((OPN .NE. UNLFEP) .AND. (OPN .NE. UNLSTP)) GO TO 12            00536500
      I=IDXLL(INTNAME,OPN)                                              00536600
C FIND INTERNAL NAME ON LOCKLIST                                        00536700
      IF (I .LE. 0) GO TO 804                                           00536800
C CANNOT FIND IT IF I .LE. 0                                            00536900
      DO 13 J=1,ITALK                                                   00537000
      IF (LLIST(2+J,I) .NE. IUCB(J)) GO TO 802                          00537100
C JUMP IF UNLOCK REQUESTED BY USER/TERMINAL WHO/WHICH DID NOT SET LOCK  00537200
   13 CONTINUE                                                          00537300
      LLIST(1,I)=-1                                                     00537400
C UNDO THE LOCK AND MARK SLOT IN UCB ARRAY EMPTY                        00537500
      GO TO 801                                                         00537600
   12 IF (TESTSE(CSI) .EQ. -1) GO TO 12                                 00537700
C-------------------------------------------------                      00537800
C  ENTER CRITICAL SECTION FOR LOCKING OUT DATUMS                        00537900
C-------------------------------------------------                      00538000
      I=IDXLL(INTNAME,OPN)                                              00538100
C GET RELATIVE LOCATION OF LOCKED DATUM IN LOCKLIST                     00538200
      IF(I .LE. 0) GO TO 14                                             00538300
C IF DATUM NOT LOCKED TO THIS OPN, GO TO 14                             00538400
```

155788

```
C NOW SEE IF DATUM FOUND ON LOCKLIST LOCKED BY THIS USER AND TERMINAL   00538500
                                                                        00538600
      DO 15 J=1,ITALK                                                   00538700
      IF (LLIST(2+J,I) .NE. IUCB(J)) GO TO 807                          00538800
   15 CONTINUE                                                          00538900
      IF((OPN .EQ. FLOCKP) .OR. (OPN .EQ. SLOCKP)) GO TO 809            00539000
   14 I=-I                                                              00539100
      IF ((OPN .NE. FLOCKP) .AND. (OPN .NE. SLOCKP)) GO TO 18           00539200
C JUMP IF NOT A LOCK OPERATION                                          00539300
      IF (I .EQ. 0) GO TO 808                                           00539400
      K1=2                                                              00539500
      IF (OPN .EQ. FLOCKP) K1=1                                         00539600
      LLIST(2,I)=K1                                                     00539700
C SET APPROPRIATE LOCK                                                  00539800
      DO 20 J=1,ITALK                                                   00539900
   20 LLIST(2+J,I)=IUCB(J)                                              00540000
C PLACE USER AND TERMINAL ID INTO LOCKLIST                              00540100
      LLIST(1,I)=INTNAME                                                00540200
C PLACE INTERNAL NAME ON LOCKLIST                                       00540300
      GO TO 801                                                         00540400
C                                                                       00540500
   18 CALL VIRTUAL(INTNAME,DATUM,OTHER,COMP)                            00540600
C VIRTUAL RETURNS IN DATUM THE VIRTUAL ADDRESS OF THE DATUM SPECIFIED   00540700
      IF (COMP .GT. 1) GO TO 810                                        00540800
C JUMP IF ERROR RETURN FROM VIRTUAL                                     00540900
      IF (OPN .EQ. STOREP) GO TO 21                                     00541000
      CALL FETCH(DATUM,RESLT,LENGTH,COMP)                               00541100
      IF (COMP .EQ. 2) GO TO 812                                        00541200
C JUMP TO 812 IF END OF DATA SET ENCOUNTERED                            00541300
      IF (COMP .GT. 1) GO TO 803                                        00541400
      CALL UNSCRAMBLE(RESLT,LENGTH,COMP,VALUE,N)                        00541500
      IF (COMP .GT. 1) GO TO 803                                        00541600
      GO TO 801                                                         00541700
   21 CALL SCRAMBLE(VALUE,LENGTH,COMP,RESLT,N)                          00541800
      IF (COMP .GT. 1) GO TO 803                                        00541900
C OPERATION PERMITTED BUT GAVE ERROR WHEN ATTEMPTED                     00542000
C                                                                       00542100
C NOW PERFORM A PHYSICAL WRITE OF N STORAGE UNITS TO THE BLOCK STARTING 00542100
C AT RESLT                                                              00542200
      CALL STORE(DATUM,RESLT,N,COMP)                                    00542300
      IF (COMP .GT. 1) GO TO 803                                        00542400
      GO TO 801                                                         00542500
   10 UCB1(II,NUCB)=INTNAME                                             00542600
      GO TO 801                                                         00542700
   11 UCB1(II,1)=-1                                                     00542800
C DETACH FORMULARY                                                      00542900
C (THIS LEAVES AN OPEN SLOT IN THE UCB TABLE)                           00543000
      GO TO 801                                                         00543100
C                                                                       00543200
  812 COMPCODE=COMPCODE+1                                               00543300
  811 COMPCODE=COMPCODE+1                                               00543400
  810 COMPCODE=COMPCODE+1                                               00543500
  809 COMPCODE=COMPCODE+1                                               00543600
  808 COMPCODE=COMPCODE+1                                               00543700
  807 COMPCODE=COMPCODE+1                                               00543800
  806 COMPCODE=COMPCODE+1                                               00543900
  805 COMPCODE=COMPCODE+1                                               00544000
  804 COMPCODE=COMPCODE+1                                               00544100
  803 COMPCODE=COMPCODE+1                                               00544200
  802 COMPCODE=COMPCODE+1                                               00544300
  801 CS1=1                                                             00544400
C -------------------------------------------------------------------   00544500
C LEAVE CRITICAL SECTION FOR LOCKING OUT DATUMS                         00544600
C -------------------------------------------------------------------   00544700
      RETURN                                                            00544800
      END                                                               00544900
```

Appendix A--FORTRAN Version of ACCESS Procedure (cont'd.)

```
      INTEGER FUNCTION IDXLL(INTNAME, OPN)
      IMPLICIT INTEGER(A-Z)
      INTEGER INTNAME,OPN
C IDXLL, GIVEN AN INTERNAL NAME INTNAM AND AN OPERATION  OPN,
C RETURNS THE RELATIVE POSITION OF INTNAM ON THE LOCKLIST IF
C IT IS LOCKED IN A MANNER AFFECTING OPERATION  OPN.  OTHERWISE,
C IDXLL RETURNS THE NEGATION OF THE FIRST EMPTY RELATIVE LOCATION
C ON THE LOCKLIST.  IF THE LOCKLIST IS FULL AND THE INTNAM/ OPN
C COMBINATION IS NOT FOUND, IDXLL RETURNS 0.
C
      COMMON/CONSTANTS/NUCB,NFORM,MAXUSERS,MAXLLIST,ITALK,
     1  FORM1,FORM2,FORM3,
     2  NEXTALL,SAMEALL,
     3  FETCHP,STOREP,UNLFEP,UNLSTP,FLOCKP,SLOCKP,ATTACHP,DETACHP
C
      COMMON/OWN1/UCB1,LLIST,CS1
      INTEGER LLIST(4,100)
      INTEGER UCB1(100,3)
      J=2
      IF((OPN .EQ. FETCHP) .OR. (OPN .EQ. UNLFEP) .OR. (OPN .EQ. FLOCKP)
     1    ) J=1
      FIRSTEMPTY=0
      IDXLL=0
      DO 1 I=1,MAXLLIST
      II=I
      K=LLIST(1,I)
      IF (K .EQ.-1) FIRSTEMPTY=I
      IF((K .EQ. INTNAME) .AND.(LLIST(2,I) .EQ. J)) GO TO 4
    1 CONTINUE
    2 IF (FIRSTEMPTY .NE. 0) IDXLL=-FIRSTEMPTY
      RETURN
    4 IDXLL=II
    5 RETURN
      END
```

```
 1 TESTSE START 0
 2 * TESTSE IS AN INTEGER FUNCTION DESIGNATOR CALLABLE FROM FORTRAN
 3 * VIA THE CALL
 4 *                        J=TESTSE(I)
 5 * I IS A VARIABLE OF TYPE INTEGER*4.  J CONTAINS, ON RETURN,
 6 * -1 ONLY IF THE CONDITION CODE WAS 1 AFTER EXECUTING THE TS OPERATION
 7 * ON I.  THE LEFTMOST BYTE OF I IS SET TO ALL ONES ON
 8 * RETURN FROM TESTSE.
 9 *
10 * THANKS TO JOHN EHRMAN FOR THE CODING OF THIS.
11 *
12       L     1,0(0,1)
13       TS    0(1)
14       BALR  0,0
15       SLL   0,3
16       SRA   0,31
17       BR    14                                          1465A16
18        END
```

APPENDIX B -- THE ACCESS PROCEDURE — "NO PARALLELISM" VERSION

This appendix presents a version of the ACCESS algorithm which can be used when no user will ever have to lock out access to a datum which ordinarily can be accessed by several users at the same time or if the installation wishes to use a method other than the one given in Section K of Chapter III to control conflicts among users competing for exclusive access to datums.

```
procedure access (info, intname, val, length, opn, compcode);
integer array info, val; integer intname, length, opn, compcode;
begin comment
```

If OPN = FETCH, VAL is set to the value of the datum represented by INTNAME.

If OPN = STORE, the value of the datum represented by INTNAME is replaced by the value in the VAL array.

If OPN = ATTACH, the formulary represented by internal name INTNAME is attached to the user and terminal described in the INFO array.

In OPN = DETACH, the formulary represented by internal name INTNAME is detached from the user and terminal described in the INFO array.

VAL is LENGTH storage elements long.

Note that a FETCH (STORE) operation will actually attempt to fetch (store) LENGTH storage elements of information. It is the responsibility of the TALK procedure to handle scrambling or unscrambling algorithms that return outputs of a different length than their inputs.

ACCESS returns the following integer completion codes in COMPCODE:

1 normal exit, no error

3 operation permitted by CONTROL procedure gave error when attempted

5 cannot handle any more User Control Blocks (would cause table overflow)

6 attempt to detach nonexistent user/terminal/formulary combination

10 error return from VIRTUAL procedure

11 operation on the datum represented by INTNAME not permitted by CONTROL procedure of the attached formulary

12 end of data set encountered by FETCH operation.

Note that by the time the user has left the ACCESS routine, the data may have been changed by another user. Note that ACCESS could be altered to allow scrambling and unscrambling to take place at external devices rather than in the central processor.

Important: ACCESS expects the following to be available to it. The installation supplies these in some way other than parameters to ACCESS (for example, as global variables in ALGOL or COMMON variables in FORTRAN) —

(1) ISTDUCB — the default User Control Block. Its length is NUCB storage units.

(2) NUCB — see (1).

(3) UCB — a list of User Control Blocks (UCBs) initialized outside ACCESS to ucb(1, 1) = -2,

ucb(i, j) = anything when ~(i=j=1)

UCB is declared as integer array (1: maxusers, 1: nucb).

(4) MAXUSERS — the maximum number of users which can be actively connected to the system at any point in time.

(5) ITALK — the length of the INFO array (which is the first parameter of ACCESS) — INFO contains information about the user and terminal which is used by ACCESS and also passed by ACCESS to procedures of the attached formulary. INFO(1) contains user identification.

ACCESS assumes that the variables FETCH, STORE, FETCHLOCK, STORELOCK,

```
UNLOCKFETCH, UNLOCKSTORE, ATTACH, and DETACH have been initialized
globally and are never changed by the installation;
integer array iucb (1:nucb), reslt (1:length);
integer i, ii, islot, j, yesno, other, n, datum;

procedure ret (i); integer i;
begin comment RET sets the completion code compcode to i and then causes
exit from the ACCESS procedure;
compcode := i; go to FIN
end ret;

compcode := 1;
comment first let's see if we recognize the user/terminal combination
in INFO;
islot := 0;
for i := 1 step 1 until maxusers do
    begin ii := i;
    if ucb [i, 1] = -2 then begin comment end of list of ucb's;
                        if islot = 0 then begin if ii ≠ maxusers then
                        ucb [ii+1, 1] := -2; go to XFER
                                        end
                                else go to PRESETUP;
                        end
    else if ucb [i, 1] = -1 then islot := ii
        comment remember this islot if vacant;
    else begin for j := 1 step 1 until italk do
```

```
                if ucb [i, j] ≠ info [j] then go to ILOOPND;
            go to SETUPPTRS
        end;
ILOOPND:
    end i loop;
if islot = 0 then ret (5); comment cannot handle any more UCBs;
PRESETUP:
ii := islot;
XFER:
for k := 1 step 1 until italk do ucb [ii, k] := info[k];
for k := = italk + 1 step 1 until nucb do ucb[ii, k] := istducb[k];
SETUPPTRS:
for i := 1 step 1 until nucb do iucb[i] := ucb[ii, i];
comment set up pointers to appropriate user control block for particular
implementation. Note well: Setting up pointers to appropriate user control
blocks is quite dependent on the particular system. For an example of one
implementation, see Appendix A;
comment We have now associated user and terminal with user control block
(representing formulary) in relative position ii of the ucb table;
if iucb[nucb] ≠ intname and opn = DETACH then ret (6);
comment attempt to detach user/terminal/formulary combination not currently
attached;
control (intname, opn, yesno, other);
if yesno > 1 then ret (11);
comment return 11 if CONTROL does not permit operation;
if opn = ATTACH then begin ucb[ii, nucb] := intname; go to FIN
                    end;
```

```
comment Note well: In many implementations, pointers to each procedure of the
formulary (obtained by having VIRTUAL transform intname into a virtual address)
might be put into the UCB upon attachment. In others, the philosophy used here
of only putting one pointer — to the formulary — into the UCB will be followed.
The decision should take into account design parameters such as implementation
language, storage available, etc.;
if opn = DETACH then begin comment detach formulary (this leaves an open
                              slot in the ucb array); ucb(ii, 1) :=-1; go to FIN
                              end;
virtual (intname, datum, other, compcode);
comment VIRTUAL returns in datum the virtual address of the datum specified;
if compcode >1 then ret (10); comment error return from VIRTUAL;
if opn = STORE then
    begin comment store operation;
    scramble (val, length, compcode, reslt, n);
    if compcode >1 then ret (3);
    comment operation permitted but gave error when attempted;
    comment now perform a physical write of n storage units to the block
    starting at reslt;
    store (datum, reslt, n, compcode);
    if compcode >1 then ret (3)
    end
else
    begin comment fetch operation;
    fetch (datum, reslt, length, compcode);
    if compcode = 2 then ret (12); comment end of data set encountered;
    if compcode >1 then ret (3);
```

```
    unscramble (reslt, length, compcode, val, n);
    if compcode > 1 then ret (3);
    end fetch operation;
FIN:
end access;
```