TO: A. Bensoussan
C. Clingen
F. J. Corbató
J. Gintell
N. Morris
R. Roach
J. Saltzer
M. Schroeder
S. Webber

FROM: V. Voydock, R. Feiertag

DATE: October 6, 1971

SUBJECT: Access Control Proposals

Enclosed is a document describing our proposals on access control, as modified by the meeting of September 8, and subsequent discussions. The proposal concerning gate lists has been tabled until Schroeder finishes documenting his plan to make it possible for a user to use more than one protected subsystem. There will be a meeting to discuss these proposals on October 13th at 2:00 p.m. in Room 511.

This document describes a number of proposed changes to the access control mechanism that have evolved over the space of several months. These include:

I. A new set of directory access attributes, and a plan to make all supervisor modules enforce access the same way.

II. Associating three ring numbers with a directory.

III. Removing the append access attribute on non-directory segments, and introducing the "maximum length" attribute.

IV. Having one set of ring brackets per segment.

V. Adding a "safety switch" to segments.

VI. Eliminating CACL's and replacing them with initial ACL's.

VII. Using extended access to implement access control on directories.

VIII. Re-defining access control for system daemons.

This section describes the newly proposed set of directory access modes and presents a plan for making all supervisor modules enforce access in the same way. In addition, a proposal to associate three ring numbers with a directory is introduced.

First we must establish some terminology. An A-operation is an operation performed on the attributes of a segment. This includes modifying its names, its ACL, its "safety switch", and its maximum length, deleting it, and listing its attributes. It also includes adding a segment to a directory. A D-operation is an operation performed on the data of a segment. This includes initiating the segment, reading it, writing it, executing it, truncating it, setting its bit count, and getting its bit count.

The major change proposed is to get rid of the "execute" directory access mode and to say that one's right to perform a D-operation on a segment is completely determined by the access information appearing on that segment's ACL. That is, to initiate a segment one needs non-null access to that segment, to get its bit count or read it, read access, to execute it, execute access, to write it, truncate it or set its bit count, write access. One's access to the directory containing the segment is not taken into account when performing a D-operation. Because a certain class of users have neeed of more security than this plan provides, we have had to modify this proposal to provide a sure means of denying access to an entire subtree of the hierarchy. This will be discussed in detail below.

D-operations on a segment are controlled by one's access to that segment (almost). A-operations are controlled by one's access to the directory containing that segment. Let us now describe the various directory access attributes:

I. null (i.e., no access attributes at all). If a user has null access on a directory, he has no access to the contents of the directory or any of its subdirectories (i.e., to the subtree of the hierarchy whose root node is that directory). In no way can he obtain information about anything in that subtree.

II. use If the user has use access on a directory, he has the potential ability to perform operations on the contents of that directory. That is, he can perform any D-operation on a given segment in that directory subject to the access that appears on that segment's ACL. For example, he can execute any segment on which he has execute permission. He can perform any A-operation on any segment in that directory subject to the other access attributes appearing on the ACL of that directory. No combination of status, modify and append access (see below) is allowed without use access.

III. status (formerly read) If a user has status access on a directory, he can list the contents of the directory and find out any and all information about the attributes of any entry in that directory, he cannot add entries or change the attributes of existing entries.

IV. modify (formerly write) If a user has modify access on a directory, he can change the attributes of existing entries. He cannot add entries or list the attributes of existing entries.

V. append If a user has append access on a directory he may add entries to that directory. He cannot add entries or list the attributes of existing entries.

There are certain combinations of directory access attributes that do not make sense. For example, as mentioned above, it does not make sense to have any combination of status, modify or append access without having use access. The file system will, therefore, not allow any of these nonsensical access combinations to be placed on a directory ACL. The following is a list of all legal access combinations: null, U, US, UA, USA, USM, USMA.

Use access has been added to provide another level of security for a certain class of users. For example, some project administrators would like to allow sharing of information among project members (including the

right of one project member to give or deny access to other project members). At the same time they would like to be sure that no project member, either accidentally, or deliberately, can give access to anyone outside that project to any of this information. They can do this by giving project members use access in the project directory and everyone else null access there. One could argue that a project administrator has to trust his project members and that they could (for instance) print out a copy of confidential information and give it to someone who has no right to have it. This is true, but in the event of an information leak, use access narrows down the possible sources of the leak. The project administrator who has used use access to limit access to information knows that the leak must have occurred outside the system. That is, by someone making a physical copy of the information rather than by someone inadvertantly or deliberately giving an unauthorized person access to the information on-line.

Now let us consider what may happen if a user tries to perform an operation on an entry whose pathname is >D1>...>Dn>E. Seven distinct error conditions (related to access control) may occur.

I. error_table_$noentry ("Entry not found")

E does not exist.

II. error_table_$no_directory ("Some directory in path specified does not exist")

One of D1, ...Dn does not exist.

III. error_table_$incorrect_access ("Incorrect access to directory containing entry")

The user does not have correct access on Dn to perform the operation.

IV. error_table_$moderr ("Incorrect access on entry")

The user does not have the correct access on E to perform the operation.

V. error_table_$safety_switch_on ("Attempt to delete segment whose safety switch is on"

The user tried to delete E and the safety switch of E is on.

VI. error_table_$null_access ("No access to subtree of hierarchy")

The user has null access on one of D1, ...,Dn.

VII. error_table_$no_info ("Insufficient access to return any information")

The user does not have enough access to be given any information.

The following flow charts describe what access checks must be made by all modules of the supervisor that manipulate segments. These checks should be made when a segment fault, incorrect access fault and undefined access fault occurs as well as by the file system primitives.

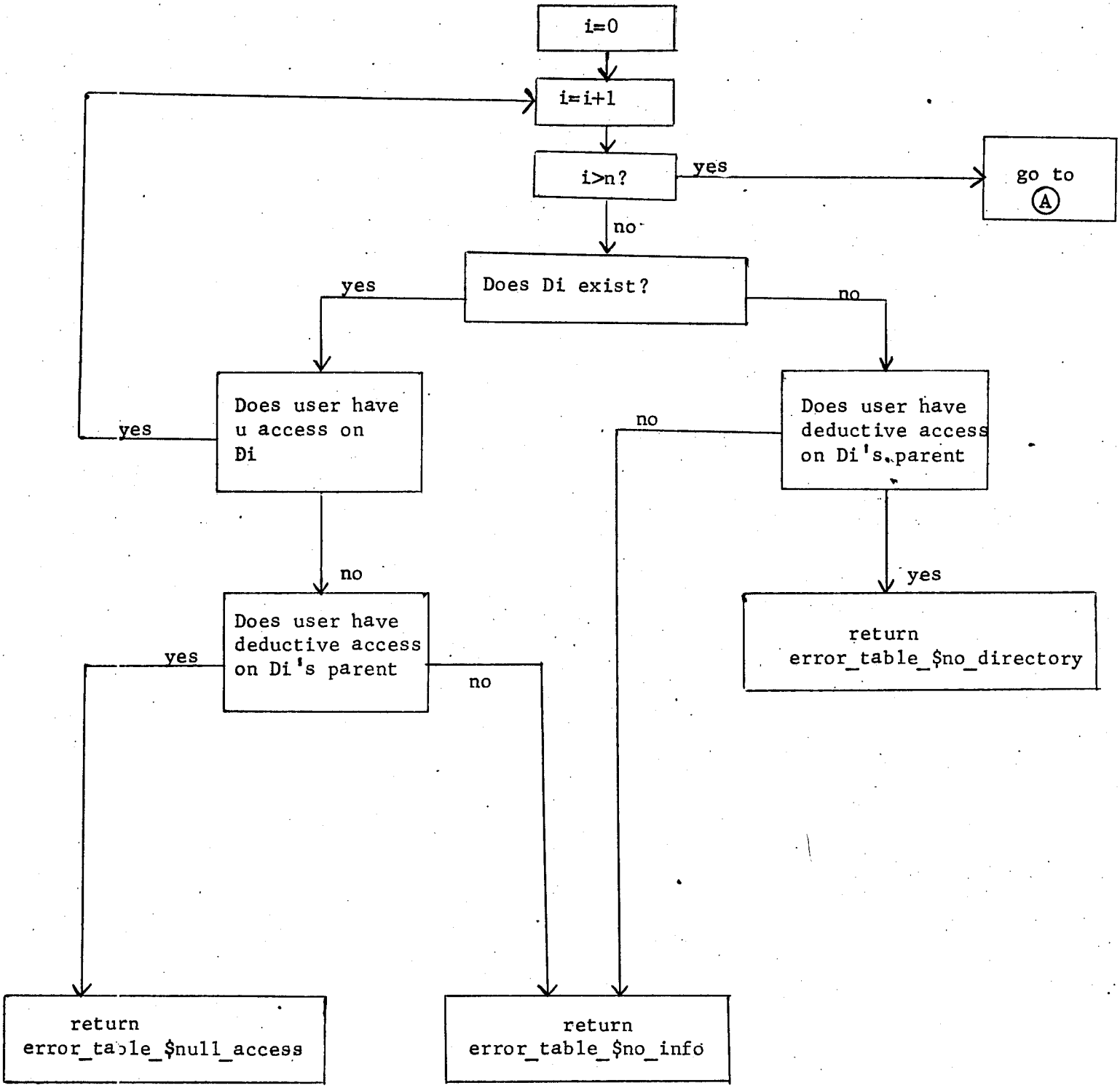
The following principles are implicit in the flow charts. The motivation for them is that they simplify the access checking mechanism and that they give away little (if any) information that couldn't be determined by experimentation.

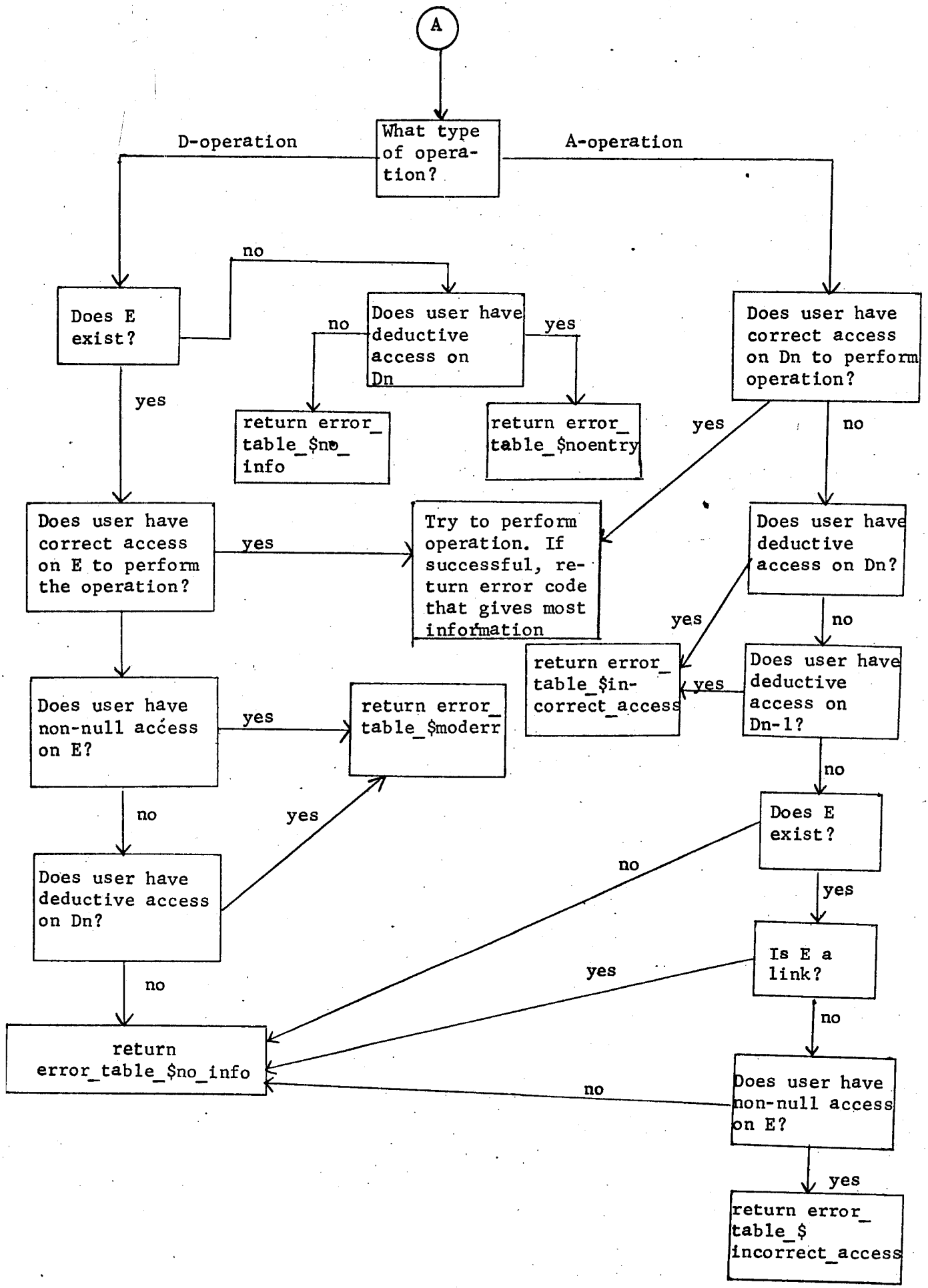
1. If one has non-null access on a segment (directory or non-directory) one has the right to know of its existence and one's effective access to it.

2. If one has deductive access on a directory, one has the right to know of the existence of particular entries in it and one's effective access to them, where deductive access is defined to be one of US, UA USA, USM, and USMA.

As an aside, note that these principles imply that if one has non-null access to a segment or deductive access on the directory containing the segment, the status primitive should admit the segment exists and return one's effective access to it -- even if one does not have status permission in the directory containing the segment.

Let us now consider the flowcharts:


i=0
i=i+1
i>n?
yes
go to
A
no
Does Di exist?
yes
no
Does user have
u access on
Di
yes
Does user have
deductive access
on Di's parent
no
no
yes
Does user have
deductive access
on Di's parent
yes
no
return
error_table_$no_directory
return
error_table_$null_access
return
error_table_$no_info


A
What type of operation?
D-operation
A-operation
Does E exist?
no
yes
Does user have deductive access on Dn
no
yes
return error_table_$no_info
return error_table_$noentry
Does user have correct access on Dn to perform operation?
yes
no
Does user have correct access on E to perform the operation?
yes
Try to perform operation. If successful, return error code that gives most information
Does user have deductive access on Dn?
yes
no
return error_table_$incorrect_access
Does user have deductive access on Dn-1?
yes
no
Does user have non-null access on E?
yes
return error_table_$moderr
no
yes
Does E exist?
no
yes
Does user have deductive access on Dn?
yes
no
Is E a link?
yes
no
return error_table_$no_info
Does user have non-null access on E?
no
yes
return error_table_$ incorrect_access

Finally, we propose that every directory have three ring numbers r1, r2 and r3 associated with it. R1 is called the modify ring and is defined to be the highest ring in which M and A access applies. R2 is called the status ring and is the highest ring in which S access applies. R3 is called the use ring and is the highest ring in which U access applies. We require that $r1 \leq r2 \leq r3$. The use ring needs to be distinct from the status ring to allow for subsystems where it is desired to give the user direct access to the data of a collection of segments in a given ring but ability to obtain information about the attributes of these segments only in a lower ring. The file manager of A. Bensoussan is one example of such a subsystem.

Now consider non-directory segments. They currently have the access attributes read, execute, write, and append. The latter attribute append, was intended to allow a process to add data to the end of a segment but not allow modification of the data already in the segment. Unfortunately, we are not currently able to implement this attribute. The append attribute is currently used to allow growing of the segment, i.e., add new pages to the end of the segment. The current use of the append attribute is not well known or well used. It is primarily used to artificially set a maximum length on a segment, a feature that should be more properly implemented by adding a maximum length attribute to a segment. Since there is currently no proper use of the append attribute it should either be deleted from ACLs or it should have no interpretation, i.e., reserved for a later proper implementation.

Besides the access attributes, segments also have sets of ring brackets. The current association of a set of ring brackets with a segment and a user has the disadvantage of being difficult to explain and visualize. With the current scheme a segment exists in different rings for different processes. A great deal of simplification is achieved by having only one set of ring brackets associated with a segment. This simplification causes no loss of functional capability because any accessing rights that can be granted by multiple sets of ring brackets on a segment can be achieved by having a procedure in a privileged ring simulate the access associated with the segment. This modification also solves the problem of what ring brackets are to be associated with a process not specified on the ACL. Clearly with one set of ring brackets, those are the only brackets that apply.

The current delete primitive requires both write permission on the segment and modify permission in the directory in order to delete a segment. This property has been used as a means of providing self-protection against accidental deletion of segments, i.e., if the segment does not have write permission, it cannot be deleted. This has the strange property of protecting object segments but not protecting data segments against

deletion. It, therefore, seems more useful to provide an attribute which allows any segment to be protected. For this purpose the "safety switch" is introduced. If the "safety switch" is on, the segment cannot be deleted. This added protection eliminates the necessity for requiring write permission on a segment in order to delete it. Therefore, the delete primitive will require modify permission in the directory and the "safety switch" being off in order to delete a segment.

The CACL is a means by which access to a group of segments can be controlled easily. Unfortunately the grouping used by the current CACL mechanism, i.e., all segments in a single directory, is not an appropriate one. It is usually not the case that all segments in a particular directory want similar access. Secondly, since the CACL is logically appended to the ACL of a segment the effect of changing a CACL upon the access to any particular segment is unclear. It depends on the contents of that segment's ACL. Thirdly, in a multiple ring situation, the rules concerning modification and use of CACLs become complex and unworkable and render the CACL useless. For these reasons the CACL is to be eliminated from Multics. The detailed arguments are given in the memo on CACLs dated June 7, 1971.

Some useful features of CACLs will be preserved. Access to large classes of segments can be modified by use of the star convention in ACL commands. Also default initial values for ACLs can be established by the use of the initial ACL.

The initial ACL is a means by which a user can specify the ACL to be added to a newly created segment in a specific directory. Each directory will contain two sets of initial ACLs, one for newly created directories and one for newly created non-directory segments. Each of these two sets will contain an initial ACL for each ring.

Each initial ACL will consist of a list of ACL entries. When a new segment is created via a call to append, the appropriate initial ACL will be found by using the type of the segment (directory or non-directory) and the current validation level. The list of ACL entries contained in this initial ACL is then used to form the ACL of the new segment. The ACL entries specified in the call to append are then added to the new ACL.

New primitives and commands will be provided to manipulate initial ACLs. Separate commands will be provided to set entries (add or change), list entries, and delete entries for both initial ACLs applying to directories and non-directory segments. The validation level at the time of the operation will determine which ring's initial ACL is involved. When an entry is added to an initial ACL it is checked to make sure the specified user id is valid. This guarantees that the initial ACL can be validly added

to a new segment with no possbility of error.

The file system currently supports a feature called extended access, the purpose of which is to allow subsystems a convenient way of specifying access attributes other than the standard attributes, on segments which those subsystems manage. Extended access is implemented simply as a set of bits in each ACL entry which the subsystem may set and interpret as it pleases; the file system does not interpret these bits in any way. Currently, extended access is used only by the message segment primitives as a means of specifying access to message segments.

Directory control is another logical candidate as an application of extended access. Currently the directory access attributes occupy the same bits as the standard segment access attributes and the directory ring brackets also occupy the same storage as the standard segment ring brackets. This duplicity of use has led to the unfortunate result of unwanted similarity between non-directory segment and directory segment access attributes. In order to allow full independence in the selection of directory access attributes, and make directory control and segment control more independent, the directory attributes should be handled separately from segment access attributes. The obvious separate mechanism is that of extended access. Directory segments would have standard access which would be rw for all users on the ACL with ring brackets of 0,0,0 on the segment and would also have some form of the SUMA directory attributes as extended access. As stated above this has the significant advantage of permitting the segment management facility to compute access in a uniform manner for all segments without having to special case directories as is presently done.

As currently implemented extended access provides extra bits for access attributes on each ACL entry. However, the proposed modification of making ring brackets a per segment rather than per ACL entry attribute means that some standard access attributes are no longer in the ACL entry. For consistency, therefore, an extended ring bracket field should be added to each branch as an extension of the standard ring bracket field in order that subsystems using extended access can treat ring brackets in a manner akin to the standard access.

Multics currently supports several service processes known as daemons. The daemon processes now includes backup, dumper, retriever, to, and translator. It is expected that sometime in the future this list will change to include backup, dumper, retiever, io, and offline segments. Each of these daemons require access to the particular part of the file system hierarchy upon which they are currently operating. This access is currently achieved by having these process's ids appear on the SPACL

and on all ACLs upon which they will have to operate. This current scheme forces users to be aware of the daemon accessing problems. This is unnecessary because the daemons are system processes and can be programmed so as to not reference segments unless they have a legitimate reason to do so.

Daemons can be split into two categories: those which perform operations at the explicit request of a user and those which perform operations without the knowledge of the user. The former category includes io, offline segment, and retriever which perform these specified operations only after receiving an explicit user request either by a command or by written form. In these cases the daemon only needs to verify that the user making the request has appropriate access to the segments in question. The daemon process itself can have access to all segments in the file system since it will, if coded properly only reference those segments implied by the explicit request and to which the requestor has appropriate access. The latter category includes backup and dumper. These daemons should, in theory, have access to all segments in the file system since they are supposed to reference all segments. If the Multics backup system were totally transparent and secure, these daemons should have free access to all segments, however, we will acknowledge that this is not the case by allowing users to inhibit the backing up of certain segments by turning off a backup attribute in the branch of that segment.

Essentially what is being proposed here is that all daemon processes potentially have full access to all segments and that the processes themselves have the responsibility for making sure they are legitimately referencing the segments. This has the advantage of isolating the facilities and services provided by these daemon processes from their implementation, i.e., that they are separate processes. The naive user should not have to be aware that the backup daemon needs access to his files or that he must grant some process called the IO Daemon access to his files in order to dprint them. The issuing of a dprint command or a retrieval request is all that should be necessary. This scheme also eliminates the need for the SPACL.

Daemon processes, for very practical reasons, should not have blanket access to all segments in the file system. This could lead to chaos should there be bugs in the implementation of a daemon process. Daemons should simply be given the right to specify their access to specific segments or perhaps reference segments through special primitives. This would minimize the potential damage in case of bugs.

The means by which daemons access to segments would be limited is as follows. Each daemon process would be granted special access properties when it is created. These properties would specify the type of access

necessary for the particular daemon to perform its functions. For example, the backup daemon needs read permission on all segments, status and use permission on all directories, plus the ability to modify the backup information in directories such as the date-time dumped. The special properties given the backup daemon process would give this process the ability to attain these access rights to all segments. Each particular daemon process is given only the access properties it needs and since the properties are granted at process creation time they are granted by the system control process which is the only process granted the right to distribute special properties, therefore keeping these special rights under tight control.

However, having these special rights is not in itself sufficient to allow the daemon processes to perform their functions. Each daemon must invoke special primitives in order to apply the access properties to specific segments. In the case of directory operations special primitives will be provided to accomplish the desired functions such as setting date-time dumped or listing the contents of a directory. These primitives will check to make sure that the specific daemon process has the necessary special access property. For operations on non-directory segments a special initiate primitive will be provided that permits processes with the appropriate access property to initiate segments which particular access rights independent of what appears on the access control list of that segment. In this manner, daemon processes will have access to only a few segments at a time rather than all segments.

These two new features: the special access properties and the special primitives, assures that daemons always have the necessary access to perform their functions, but only that access which is necessary and that the potential damage in case of error is minimized because the daemon processes have to make specific special calls to gain these access rights.