TO: Distribution

FROM: V. L. Voydock, R. J. Feiertag

DATE: March 2, 1972

SUBJECT: Modifications to Access Control

This document describes a number of changes that are being made to access control mechanism. These include:

I. Removing the execute access attribute on directories and renaming (for clarity) the other directory access attributes.

II. Returning better access control related status codes.

III. Associating two ring numbers with a directory.

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

V. Having one set of ring brackets per segment.

VI. Adding a "safety switch" to segments.

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

VIII. Eliminating the SPACL and automatically placing a "*.SysDaemon.*" entry on the ACL of a segment when that segment is created.

Let us first 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 C-operation is an operation performed on the contents of a segment. This includes initiating the segment, reading it, writing it, executing it, truncating it, setting and getting its bit count and call limiter.

The major change proposed is to get rid of the "execute" directory access mode and to say that one's right to perform a C-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 gets its bit count, get its call limiter or read it, read access, to execute it, execute access, to write it, truncate it, set its call limiter or set its bit count, write access. One's access to the directory containing the segment is not taken into account when performing a C-operation.

In summary, C-operations on a segment are entirely controlled by one's access to that segment, and A-operations are entirely controlled by one's access to the directory containing that segment. Let us now describe the various directory access attributes:

I. 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.

II. 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.

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

The access combinations M and MA do not make sense (at least no one has yet found a use for them). The file system, therefore, will not allow them to be placed on an ACL. If a legitimate use for either of these combinations is found the restriction can easily be removed.

Now let us consider what may happen if a user tries to perform an operation on an entry whose pathname is >D1>...>Dn>E. Six 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_$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 occurs as well as by the file system primitives.

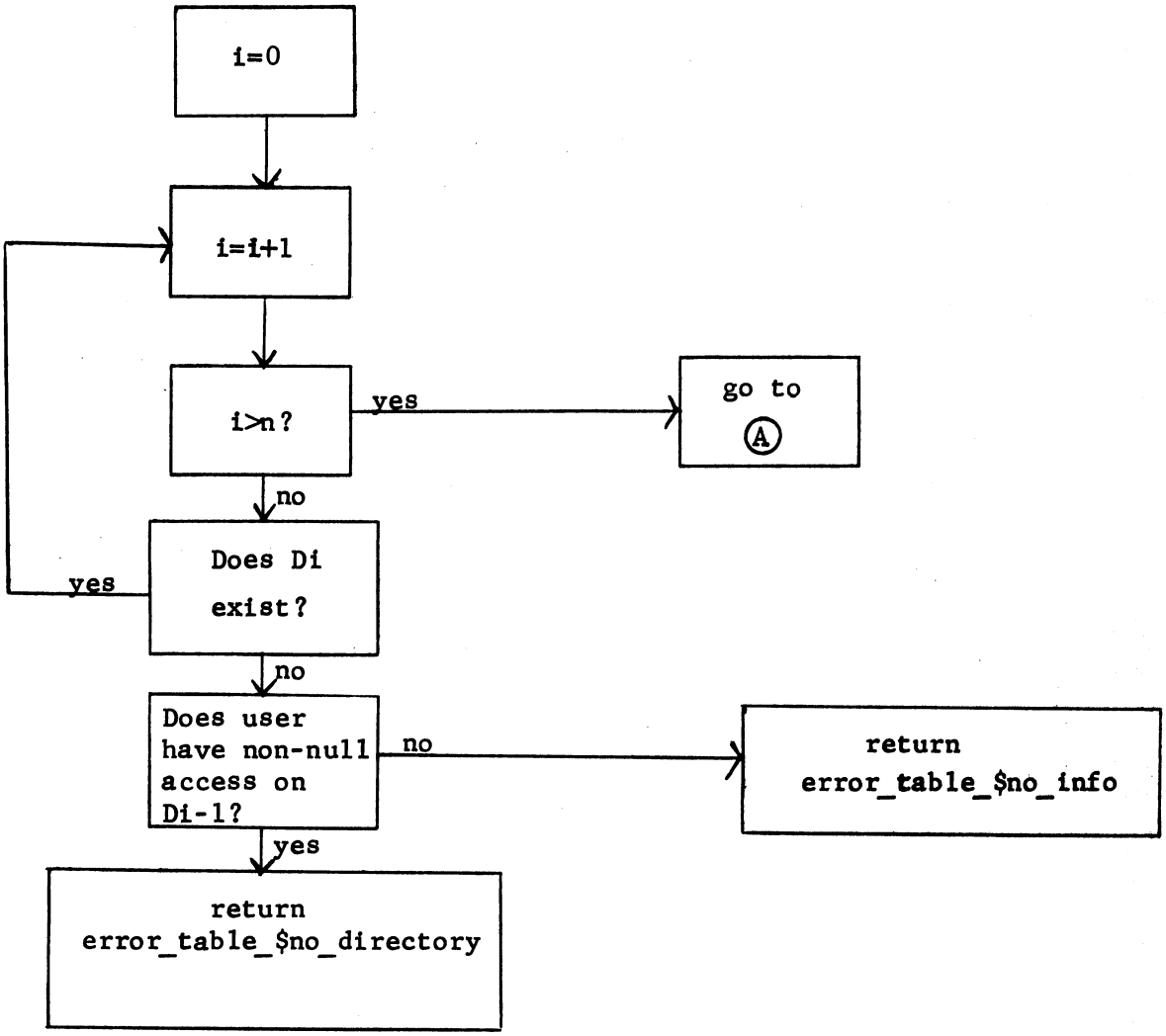
The following principles are implicit in the flow charts. The motivation for them is that they simplify the access checking mechanism and that they give away no information that couldn't be determined by experimentation.

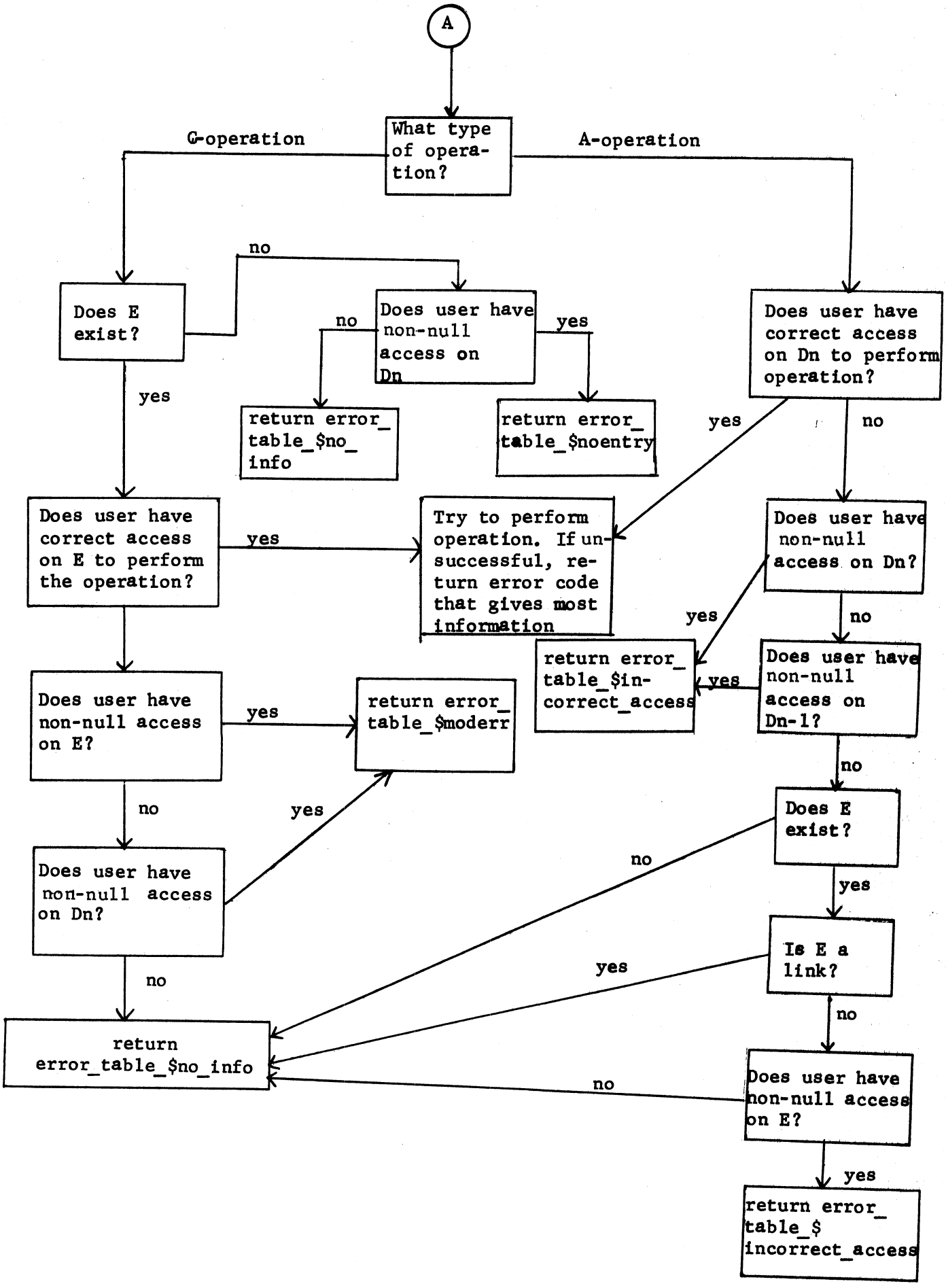
1. If one has non-null access on a segment (directory or non-directory) one has the right to know of its existence and one's effective access to it.

2. If one has non-null 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.

As an aside, note that these principles imply that if one has non-null access to a segment or non-null 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 (we are performing an operation on a segment whose pathname is >D1>...>Dn>E):


i=0
i=i+1
i>n?
yes
go to
A
no
Does Di
exist?
yes
no
Does user
have non-null
access on
Di-1?
no
return
error_table_$no_info
yes
return
error_table_$no_directory


A
What type of operation?
G-operation
A-operation
Does E exist?
no
yes
Does user have non-null 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 unsuccessful, return error code that gives most information
Does user have non-null access on Dn?
yes
no
return error_table_$incorrect_access
Does user have non-null access on Dn-1?
yes
no
Does user have non-null access on E?
yes
return error_table_$moderr
no
yes
Does E exist?
no
yes
Does user have non-null 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 two ring numbers r1, and r2 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 appplies. We require that r1≤r2.

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 will be deleted from ACLs.

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.

Multics now supports several service processes termed daemons. In order to properly perform their designated function these daemons sometimes require access to user segments. This is currently accomplished by a combination of two special mechanisms. The Special Access Control List (SPACL) is the means by which daemon processes gain access rights to any directory in the hierarchy. Secondly, whenever a directory is created the CACL created for that directory is initialized to contain an entry granting access to all segments in the directory to the daemons. The SPACL is to be eliminated because it grants full access by daemons to directories and this access cannot be overridden. Users, therefore have no means of denying access to their segments to processes appearing on the SPACL. Also the SPACL is not currently visible to users, thereby making it impossible to determine who has access to a given segment. Since the CACL is being eliminated for reasons given earlier, that means of granting access to daemons is no longer useable.

Users should be able to exercise maximal control over access to their segments. Maximal control enables users to deny access to any process or procedure in which they do not have complete faith. This control should not only include access by other user's processes, but also system processes and procedures even if these

system processes or procedures are essential for normal service. A user having critical information in the Multics hierarchy should be able to deny access to those procedures which he has not been able to validate, however, in doing so he must accept the risk that by denying these procedures access to his segments he may be impairing the systems ability to properly handle these segments. For example, if a user denies backup access to his segments they will not be backed up.

With regard to daemon access we can follow this policy by having these daemon processes derive this access from the normal access mechanism. In order for the daemon to access a segment, that daemon must have appropriate access on the necessary ACLs. In order to deny a daemon access to a segment one simply declines to give it access. However, this would impose a hardship on the majority of users who want the daemons to have access by forcing them to be aware of daemon operation and making them give the daemon access.

In order to transfer the burden of effort from the many to the few, two special actions are to be taken. First, when a segment is created an entry for *.SysDaemon.* should be placed on the ACL for that segment, and the initial ACL and ACL specified in the append call should be added. For non-directory segments this daemon ACL entry will permit rw access and for directory segments sma access. Secondly, those ACL manipulating entries that entirely replace ACLs will contain a switch in their calling sequences. One setting of this switch will cause the special daemon entry to be logically added to the ACL before the specified entries. The other setting of the switch will perform the normal function. These two measures will assure that no user will naively remove daemon access but does allow those users who wish to restrict access to do so simply.

Because of the special treatment given those processes in the SysDaemon project, only those processes which truly must be daemons should be members of this project and their function should be properly segregated and identified in order that users may selectively choose those procedures they wish to trust and deny access to others by specifically denying those daemon processes access to their segments. For this reason human users, such as Repair, registered as part of the SysDaemon project should be moved to some other project. Also only those functions specified for a particular daemon process should be performed by that process. Currently the following daemon processes should be registered:

Backup
Retrieve
IO
Dumper