TO: A. Bensoussan
C. T. Clingen
F. J. Corbató
J. W. Gintell
N. I. Morris
J. H. Saltzer
M. D. Schroeder
S. Webber

FROM: V. Voydock, R. Feiertag

DATE: July 26, 1971

SUBJECT: Access Control on Directories

Attached is yet another document on access control. Please try to read it before the meeting of Tuesday, July 27, at 3:00 p.m. I apologize for this last minute distribution but the ideas presented here did not jell until last Friday.

The purpose of this document is to present a plan for making all the file system primitives enforce access control in the same way and, at the same time, clear up some confusion about the meaning, with respect to directories, of the various access modes.

The first step is to list the various modes and their interpretation:

I. use (formerly execute) If a user has use access to a directory, he may "use" any program in that directory (that he has the correct access to). This includes executing the segment (if he has execute permission on it), reading it (if he has read permission), deleting it (if he has delete permission), truncating it and setting its bit count (if he has write permission). He can list specific entries but the only information he is given is the fact that the entry exists and his access with respect to that entry. He may not use the star convention. He cannot add entries or change the attributes of existing entries.

II. 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 any entry in that directory. He cannot "use" programs in that directory, add entries or change the attributes of existing entries.

III. modify (formerly write) If a user has modify access to a directory, he can change the attributes of existing entries in that directory. He cannot list the directory (not even specific entries). He cannot "use"

or add entries. Modify permission without status or use permission is not particularly useful and it complicates the access control mechanisms (especially error handling). Therefore, the file system will not allow this case to occur.

IV. append If a user has append access to a directory, he can add entries to the directory. He cannot list the directory (not even specific entries). He cannot modify or "use" existing entries. Append permission without status or use permission is not particularly useful and it complicates the access control mechanisms (especially error handling). Therefore, the file system will not allow this case to occur.

We come now to a concept which has caused a lot of confusion in the past. We would like to have the ability to allow someone to work in a directory even though he has no access to the directory containing this directory. For example, if one had a project in which one member was not allowed to know anything about any other member, one would like to give each member SUMA access to his own home directory and "null" access to his project directory. By themselves, rules I-IV above do not allow this, since null access to the project directory implies that the user does not have the right to know that a given entry in the project directory (including his own home directory) exists. Thus, he cannot be allowed to access segments in his home directory for the very act of accessing these segments tells him that his home directory exists.

In the current system this problem is solved by a mechanism which I will call the rule of implied access. It is stated as follows: "If a user has non-null access on a directory, he is allowed to operate in that directory regardless of his access to any superior directories". That is, the fact that he has non-null access on a directory implies that he knows it exists. The disadvantage of this mechanism is that it makes it impossible to deny someone access to a subtree of the hierarchy without denying him access to every directory in that subtree. One solution to this problem is to add another access mode P (for "pass through") to directories. P permission on a directory allows a user to do nothing in that directory, but does allow him to operate in directories inferior to that directory. Thus, in order to operate in a directory, the user must have P or U permission on all of its ancestors. To cut off access to a subtree of the hierarchy one merely puts null access on the root node of that subtree.

Let us now assume that the user wants to perform some operation on an entry E in a directory D. If he does not have P or U permission on all of D's ancestors he cannot operate in D so the file system returns the error_table_$noaccess error code (see below). Otherwise, four distinct error conditions (related to access control) may occur in trying to perform the operation on E.

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

E does not exist and the user has status or use permission in D.

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

The user has the correct access in D to perform the operation, but does not have correct access on E.

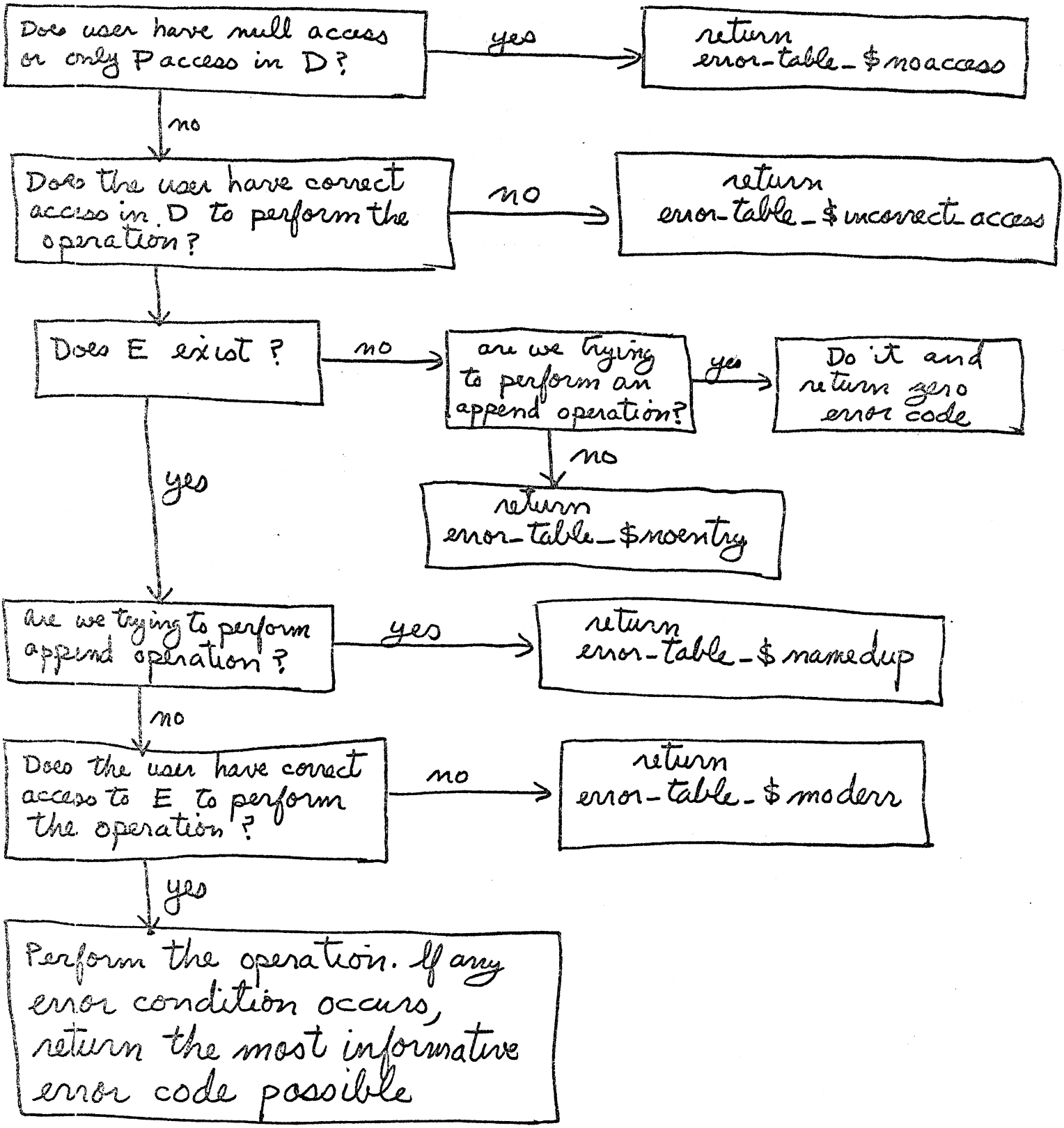
III. error_table_$incorrect_access ("Incorrect access to directory containing entry")

The user does not have the correct access on D to perform the operation, but does have status or use permission on D.

IV. Error_table_$noaccess ("No access to subtree of hierarchy")

The user has null access in D or only P access in D.

The following flow chart will indicate that these codes are both necessary and sufficient and will at the same time describe what access checks all file system primitives must make when trying to perform an operation on an entry E in a directory D.


Does user have null access or only P access in D?
yes
return error_table_$noaccess
no
Does the user have correct access in D to perform the operation?
no
return error_table_$uncorrect_access
Does E exist?
no
are we trying to perform an append operation?
yes
Do it and return zero error code
no
return error_table_$noentry
yes
Are we trying to perform append operation?
yes
return error_table_$namedup
no
Does the user have correct access to E to perform the operation?
no
return error_table_$moderr
yes
Perform the operation. If any error condition occurs, return the most informative error code possible