TO: C. T. Clingen
F. J. Corbató
J. W. Gintell
A. Kobziar
M. Meer
N. Morris
R. Roach
✔J. H. Saltzer
M. Schroeder
E. Stone
D. Vinograd
M. Weaver
S. Webber

FROM: R. Feiertag, V. Voydock

DATE: October 29, 1971

SUBJECT: New Directory Format

This document presents ideas concerning a new directory format It is somewhat disorganized due to a severe time constraint for its preparation. The first few paragraphs emphasize and motivate the important conceptual changes in the directory format change and the rest of the document describes the change in detail.

The most important significant conceptual change is in the elimination of user codes. The current implementation of ACLs imposes an arbitrary upper limit on the number of users that may be registered (by registered is meant that a user is granted a unique access identification) on the system and also permits any other user to discover if that user is registered. The first problem can be alleviated by increasing the size of the user code table but this does not solve the problem. It is proposed that ACLs contain the character string identifier of a user rather than the user code. This technique solves both problems related to user codes. It also simplifies much of the implementation of the system having to do with maintaining the user code table.

The use of character strings rather than user codes in ACLs adds some cost to the maintenance and use of ACLs. The extra space required can be minimized by keeping the character strings associated with ACLs of a particular directory in a common pool in the directory and have the ACL entries contain only relative pointer to the character strings. Since most ACLs in any given directory have most of the access names in common this should considerably reduce the storage needed. Some amount of additional time will be necessary to perform character string comparisons rather than numerical comparisons. However, with clever coding, this difference can be made negligible.

The second conceptual change is switching to variable length file maps which will serve to save space in directories since most file maps are small. This is especially important if we are to implement 256K segments.

The third significant change is in allowing variable length hash tables. This will permit most directories to consume less storage and will allow a directory to contain a greater number of names. Note that although the directory format is being changed, some of the features that make use of the new format will not be implemented at the time of the change. It is intended that only those parts of the system that must be modified to perform the reformatting be changed initially and that other features which the reformatting allows be done as time permits. An example of a feature not to be done initially is variable size hash tables.

This section describes the differences between the current directory format and the proposed new directory format. The description is divided into three parts: deletions, changes, and additions. No attempt is made to discuss how the necessary changes are to be accomplished, that will be the subject of another document.

As of this writing some issues have not been resolved and therefore will not be discussed. These issues have to do with the allocation of hash tables and the inclusion of metering statistics for directories and segments. A supplemental document will be issued when these issues have been resolved.

Deletions

1. All deletions from directory format have to do with items that will be obsoleted by the arrival of page multilevel. The deletions are:
   a) per process master device limit -- the number of records of segments in the process directory that go on the drum.
   b) master limit switch -- indicates if this segment goes on the drum.
   c) move device id -- indicates to which device the segment is to be moved.

Changes

1. The most significant change to directory reformat will be in access control lists. We plan to eliminate the user code table and in its place store the actual character strings representing access capabilities, in the directory. In order to save space the names will not be stored in the ACL entries, but in a common pool for the directory, and each ACL entry will contain relative pointers to the names. The common pool will consist of two threaded lists (threaded forward and backwards) of names. One list will be for person names and one for project names. With each name will be a count of the number of ACL entries that refer to this name. When this count becomes zero the name is deleted from the pool. Since it is important that names in the pool not be prematurely deleted (i.e., not deleted while some ACLs still refer to it), it is kept redundantly. Relative pointers to the beginning and end of the two lists will be kept in the header.

ACL entries will change considerably. ACL entries will be forward and backward threaded and will contain a relative pointer to the person name (in the person name pool), a relative pointer to the project name (in the project name pool), a tag, the standard access bits, and the extended access bits.

Each branch will contain a relative pointer to the beginning and end of the ACL and a count of the number of entries in the ACL. The purpose of the count and the end pointer is to aid salvaging. Branches will also contain the standard and extended ring brackets for the segment.

2. Due to elimination of user codes, the representation of the author of a branch will have to be changed. The author will consist relative pointers to the person name and project name, and the tag.

3. File maps for segments will be of varying sizes instead of one fixed size. The sizes to be used are 4, 16, 64, and 256 device addresses. The branch will contain, in place of the file map itself, a relative pointer to the file map and its size. The file map itself will be allocated within the directory.

4. Path names will also be allocated instead of being included in the branch. The branch will therefore contain a relative pointer to the path name, the length of the path name, and the size of the allocated area containing the path name.

5. The current branch count in the directory header will be divided into a non-directory branch count and a directory branch count.

Additions

1. Initial ACLs will consist of threaded lists of ACL entries. The directory header will contain 16 relative pointers to the beginning and end of such ACLs to be associated with directories for the 8 rings, and the other 8 will

be for non-directory branches. There will also be a date time used, date time modified, and date time dumped for all the initial ACLs in a directory and a relative pointer to backup information. The backup info pointer is for later use when additional information is kept by backup.

2. A relative pointer to backup information for the CACL will be added. It is hoped that the CACL will be eliminated before backup is upgraded to make use of the backup info pointer and therefore it will never be used. It is included just in case.

3. In order to help the pre-page, post-perge mechanisms it is useful to identify those segments which are per-process, e.g., stacks and combined linkage section. For this purpose a switch will be provided in the directory header indicating that this directory contains per process segments. A switch will also be provided in each branch indicating that it is a per process segment.

4. For the convenience of the salvager each branch will contain a relative pointer to the end of the names list.

5. Each branch will contain a backup info pointer that will eventually point to additional backup information.

6. Each branch will contain a safety switch (helps prevent accidental deletion of the segment), a maximum length, an off line segment indicator (indicates segment is off line), an entry point count (call limiter), and an entry point count enable switch (indicates entry point count is enabled).

The attached declarations define the exact structure of the proposed directory format at its current state of development.

Supplement:

In order to allow a directory to contain a greater number of names and in order to allow the majority of directories to consume less storage, hash tables will be of varying size. When a directory is created its hash table will be created large enough to efficiently contain a few entries. When the hash table becomes significantly full the directory will be restructured to contain a larger hash table. To accomplish this nothing has to be added to the directory header. In fact the current relative pointer to the beginning of the hash table is to be deleted because the hash table always immediately follows the header.

In order that the system can easily restrict the depth of the hierarchy, the depth in the hierarchy of a directory will be kept in the header. It is necessary to limit the hierarchy depth because parts of the file system cannot support an arbitrary depth.