To: Distribution

From: Peter Haber

Date: June 27, 1974

Subject: Redefinition of "*", "." and Missing Components (B-086)

The purpose of this document is to propose a redefinition of the way access control commands handle the "*" and "." characters and to propose a new common subroutine to implement this definition.

DESIGN SECTION

Current Definition Of "*", "." and Missing Components

The current meaning of these symbols to the access control commands varies with the specific command. To the ACL listing commands (listacl, message_segment_listacl), the "*" or "." or a missing component signify "any instance of". To the access manipulating commands (setacl, message_segment_setacl, deleteacl, message_segment_deleteacl), these symbols signify the literal character "*". For example, assume a segment, foo, with ACL entries a follow:

```
     rewa  A.Multics.*
     re    B.Multics.*
     re    *.Multics.*
```

The command

```
     listacl foo  *.Multics.*
```

or

```
     listacl foo  .Multics
```

would result in a list of the three entries shown, whereas the command

```
     setacl foo r *.Multics.*
```

or

```
     setacl foo r  .Multics
```

---

would result in the setting of the third ACl entry, the literal "*.Multics.*" only, and the command

```
     deleteacl foo  *.Multics.*
```

or

```
     deleteacl foo  .Multics
```

would result in the deletion of the third ACL entry only.

Proposed Definition of "*", ".", and Missing Components

For the purpose of increased consistency, the following scheme is recommended.

1) The star character will always signify the literal "*".
2) A missing component without a separating "." will always signify a literal "*".
3) A missing component with a separating "." will always signify "any instance of".

For example,

```
     command foo  *.Multics
```

means perform the command for any ACL entry with first component "*", second component "Multics" and third component "*".

```
     command foo  .Multics.
```

means perform the command for any ACL entry of foo with any first component, second component of "Multics" and any third component.

IMPLEMENTATION SECTION

The purpose of this section is to describe a primitive the implementation of which will simplify access control commands.

Current Scheme of Obtaining ACL Information

An acl manipulating command determines matches between an acl structure and user arguments in the following manner.

1) Perform a listacl of the segment in question.

2) For each access_name argument:

a) "expand" the argument by a subroutine call to determine its type, and then perform the expansion: e.g., argument = ".Multics." call to subroutine, find type = 2 therefore, expanded argument = "*"||argument||"*" = "*.Multics.*"

b) call another subroutine with a pointer to the acl list and the expanded argument. This second subroutine turns on bits in an argument bit string to indicate which acl entries match the expanded argument.

The proposed replacement subroutine allows the same information to be gathered by a single call with matching names returned in a list structure.

Proposed Replacement

Name find_common_acl_names_

This subroutines is used to make correlations between user arguments and names on the acl of a given segment.

Entry find_common_acl_names_init

This entry is called first to initialize internal data.

Usage

```
dcl finc_common_acl_names_$find_common_acl_names_init (char (*),
char (*), fixed bin, fixed bin, ptr fixed bin, fixed bin (35)):

call find_common_acl_names_$find_common_acl_names_init (dn, en,
type, ex_acl_type, aclp, acl_count, code);
```

1) dn

is the directory portion of the pathname of the segment in question (Input).

2) en

is the entry portion of the pathname of the segment in question (Input).

3) type

indicates the type of the segment in question
1 => segment
2 => multisegment file

3 => directory
4 => library segment
5 => extended access segment (Input)

4) ex_acl_type

indicates the type of extended access segment in question. (Unused if type ^= 5)
1 => queue message segment
2 => mailbox message segment
3 => vfo segment (Input).

5) aclp

is a pointer to the allocated acl structure (Output).

6) acl_count

is the number of acl entries currently associated with the segment (Output).

7) code

is an error code (Output).

<u>Entry</u> find_common_acl_names_close

This entry is called to free data allocated by the initialization entry. It need not be called but will save the caller space in his free segment between initialization calls if used.

<u>Usage</u>

```
dcl find_common_acl_names_$
find_common_acl_names_close entry (ptr, fixed bin (35));

call find_common_acl_names_$
find_common_acl_names_close (aclp, code);
```

1) aclp

is a pointer to the allocated acl structure (Input).

2) code

is an error code (Output).

<u>Examples of Use</u>

Using foo again, assume the user types

```
          la foo B.  .Multics
```

the command need only call the initiate entry with the expanded pathname (>dir>foo) then call the main entry in a loop with the remaining arguments

The first call would cause

```
     count <- 1
     names_ptr -> "B.Multics.*"
     missing_component <- "1" b
     already_used <- "0" b
```

The second call would cause

```
     count <- 2
     names_ptr -> "A.Multics.*
                  "*.Multics.*"
     missing_component <- "1" b
     already_used  <-  "1"  b  (Indicates match to previously returned
     "B.Multics.*").
```

After each call, the command places the returned names into an appropriate acl structure. When the argument parsing is done, the command calls the appropriate acl primitive with a pointer to the acl structure.

Note that since no ACL entry is returned more than once the command need not concern itself with duplicate entries. This functionality has the implication that the order in which arguments are processed is important; using foo again.

```
     sa foo re A rew .Multics
```

"A.Multics.*" matches either of the arguments "A" or ".Multics". Thus if the arguments are parsed from left to right the ACL entry for A.Multics will be set to "re", whereas if the arguments are parsed from right to left, the ACL entry for A.Multics.* will be set to "rew". The second interpretation is the correct one, and thus should be used by the access control commands.

Please send any comments to John Gintell.