To: Distribution

From: Steve Webber

Date: 04/29/75

Subject: Proposed Redefinition of the Copy Switch

This MTB proposes redefinition of the copy switch item in a directory branch. The prime reasons for wanting to do this are:

1. to make a more consistent and simpler mechanism available to users,

2. to simplify the supervisor, and

3. as a part of the implementation of copy_on_write.

Currently the copy switch is used by the initiate primitives to provide a pointer to a copy of a segment (which is potentially nonshareable) rather than to the segment itself. This means that the supervisor must do a good deal of work (in ring 0) implicitly (such as create a segment in the process directory, make it known, initiate a reference name, etc.). This work would better be done in the user ring either implicitly as in response to a copy_on_write fault or explicitly as when a user initiates a segment so that he gets a copy regardless of the setting of the branch item.

The proposal is to:

1. implement copy_on_write in the user ring with system software, and

2. control when a copy_on_write is to take place with the branch item "copy_switch".

The copy_on_write mechanism is simple and would work as follows:

1. If a potentially copiable segment is initiated, a pointer to the original segment is returned, even if the copy switch is ON.

2. If an attempt is made to write into the original and the user does not have write permission to the segment,

and the copy switch is ON the following actions are taken:

A. Create a segment in the process directory with the name "!unique...etc.copy_of..." Give the calling process REW access to the segment.

B. Make this segment known.

C. Copy the original segment into this segment.

D. Make the original segment unknown but reserve its segment number.

E. Make the copy unknown.

F. Make the copy known with the reserved segment number.

No action is taken on reference names. (It is assumed that reference names have already been dissociated from KSTE's.) Hence, the reference names which were associated with the original segment are now associated with the copy as the copy has the original segment's segment number.

It may be worth the effort to create an address space manager primitive to perform actions D, E, and F above in a single call.

The copy_on_write handler which performs the above tasks would be invoked when a "no_write_permission" fault occurs. The signal_ program will special case this before searching the stack. If the segment does not have the copy switch ON, no_write_permission is signalled in the usual way. If the copy_switch is ON, the copy is created, etc. and the fault is restarted immediately without searching the stack.

Clearly there must be a mechanism for allowing users to take whatever action they want to--possibly to ignore copy_on_write events. This can be done today by the user replacing the signal pointer in the stack header. Another possibility is proposed in an upcoming MTB on "Handling System Conditions".

This entire change is incompatible and users will need to be told about it in advance. It addition, a consistent replacement must be provided which is as similar to what we have today as is possible. The new actions taken by the hardcore primitives are proposed below:

1.	hcs_$initiate	This primitive will not create a copy as it does today. Indeed, the primitive will not even look at the copy switch in the branch. Since a value of 2 to the current copy_ctl_switch parameter will not be meaningful, specifying this value will result in failure to initiate as reflected by a new status code. The values of 0 and 1 will be allowed and handled, but no action will be taken until (if ever) a copy-on-write fault occurs.
2.	hcs_$initiate_count	This primitive will act exactly as the hcs_$initiate primitive does with respect to the copy switch.
3.	hcs_$delentry_file (_seg)	These primitives will look at the copy switch and treat the copy switch exactly as it treats the safety switch, i.e., an attempt to delete the segment will fail as long as the copy switch is ON.
4.	hcs_$status (etc.)	No change
5.	hcs_$append (etc.)	No change.
6.	hcs_$add_acl_entries (etc.)	No change.
7.	hcs_$set_bc (etc.)	No change.
8.	hcs_$fs_move_file (_sg)	No change.
9.	hcs_$terminate_file	No change.
10.	hcs_$truncate_file	If the copy switch is ON, take no action. If the copy switch is OFF, truncate as usual.
11.	hcs_$truncate_seg	If the copy switch is ON, cause the effect of a copy_on_write (i.e., if no write permission, create a copy with the same segment number) to occur and truncate the copy. If the copy switch is OFF, truncate the

segment.

These mappings provide a consistent mapping of all potential uses of the copy switch known to me except the initiate calls with a value of 2 specified in the copy_ctl_switch parameters. This effect can easily be duplicated by straight forward user-ring code.

Note that segments created as copies of other segments will, in general, not have the copy switch ON and will be writeable. Hence, it is very unlikely for a copy_on_write fault to occur on one of these segments.

A problem arises when a program of today initiates a segment known to have the copy switch ON. This program can depend on the fact that the pointer returned to him points to a copy and hence actions such as truncate and delete will not have any effect on the original. With the new proposal, however, the returned pointer will point to the original until (if ever) an attempt is made to write into it. Hence, such programs, if they never do modify the original, will perform their cleanup actions intended for the copy on the original. This is why the truncate and delete primitives will be changed to treat the copy switch specially. An incompatible problem arises here if no modifications are performed as an error code will be returned when an attempt to delete the original is made. Note, however, that all known uses of the copy switch work with no change in behavior as modifications are always done (that is why the copy switch is ON).

It would probably be useful, as noted in MTB-169, to issue warnings when 1) the copy switch is set ON for a segment with write permission granted to some user, and 2) when write permission is granted to a segment whose copy switch is ON.