To: Distribution

From: T. H. Van Vleck

Date: June 26, 1979

Subject: Adding a "Property List" to the Branch

INTRODUCTION

From time to time there have been proposals to add data items to the directory branch for the convenience of application subsystems. Although several of these proposals have been of some interest, we have never been able to justify the effort required to modify the structure of directories on the basis of the improvement to a single application; and the difficulty of designing an extension which would satisfy the conflicting needs of more than one subsystem also led us to defer consideration of such an extension. This memorandum proposes that we now implement a mechanism which appears to satisfy all known needs in a straightforward way.

PROPOSAL

Briefly, the proposal is that an optional item be kept by the file system for each branch, called the property list. This item is a list of pairs associated with the segment or directory. Each pair contains a character-string name and a value associated with the name. No name can appear twice on a branch's list.

HISTORY

The original proposal was set forth in MTB-210. This proposal differs only in a few details.

The design review of MTB-210 raised the following issues:

1. Access control - Several different proposals were made concerning how access conttrol to the property list should be provided. These included

   a) Treat the property list as if it was the contents of the segment. Allow writing if user has "w" and listing/reading if user has "r".

   b) Treat the property list as a segment attribute

controlled by the directory mode.

c) Mixtures of a and b.

d) Provide a new ACL for property lists.

No clear consensus about the optimum method was reached.

2. Radical proposals - additional time was spent discussing other proposals.

a) Don't implement property list at all. It was argued that the benefits weren't worth the additional complexity. This is especially true if some complicated access control mechanism is the only one acceptable.

b) Implement "segmentitos" and "directlets" instead.

c) Simulate the property list in ring 4.

None of these ideas seemed to gain wide support either.

We decided that more thought was needed, and adjourned. Since that time, several people suggested supporting a "mixed" access control strategy in which read/write is controlled by segment mode and add/delete by directory mode.

DETAILED PROPOSAL

Structure of the Property List

The pairs on a property list are allocated in the directory just as ACL entries are, but no attempt is made to common the space if more than one branch has the same name-value pair. The branch contains 18-bit forward and backward relative pointers to the property list (if any). The property list is threaded two ways, just like an ACL, so that the salvager can repair it.

The property name may be up to 32 characters long. The value of a property is a character string from 0 to 1024 characters long.

Access control to the property list will be the same as access control restrictions currently enforced on the contents of a segment. That is, write access on the segment itself is required, and status on the containing directory is not required. The rationale for this choice is that it is simplest, and obeys the principle of least privilege. If one has a segment to which he lacks w access, but can modify the segment's parent directory, then the property list can be modified by a two-step procedure. If, on the other hand, one has w access on a segment but not the

permission to modify its directory, this proposal permits software to function the same whether or not a property with the given name exists on the segment.

Directories may also have property lists: m access on the directory permits modification, s permits reading. Links may not have property lists.

Primitives_for_Manipulating_Property_List

Hardcore primitives are provided for the maintenance of the property list, so that application programs can add or delete properties, change the value of items on the property list, and so forth. In the examples below, the "pointer" form of calls is shown. Similar calls which accept dirname and ename will also be provided.

All of the primitives which manipulate single properties handle property values as character strings, although the supervisor does not enforce the requirement that the strings contain ASCII.

HCS_$PUT_PROP

This entry puts a name-value pair on the property list of a branch. If a property with this name existed already, its value is replaced and the old value returned. Otherwise a new entry is made.

```
dcl hcs_$put_prop entry (ptr, char(32), char(*),
    char(*), fixed bin(35));

call hcs_$put_prop (p, name, newv, oldv, code);
```

If the property did not exist before, oldv will be null and code will be error_table_$created_property.

HCS_$DEL_PROP

This entry deletes a property.

```
dcl hcs_$del_prop entry (ptr, char(32), char(*), fixed
bin(35));

call hcs_$del_prop (p, name, oldv, code);
```

The old value is also returned.

HCS_$GET_PROP

This entry searches the property list for a given property.

```
dcl hcs_$get_prop entry (ptr, char(32), char(*), fixed
bin(35));

call hcs_$get_prop (p, name, oldv, code);
```

If the property does not exist, oldv will be null and code will be error_table_$no_property.

HCS_$PUT_PROP_COND

This entry does a conditional put_prop operation. It is indivisible with respect to other put_prop operations and so can be used to manipulate "semaphore" variables.

```
dcl hcs_$put_prop_cond entry (ptr, char(32), char(*),
    char(*), char(*), fixed bin(35));

call hcs_$put_prop_cond (p, name, testv, newv, oldv, code);
```

If the entry has a property named "name" with a value which matches the test value, then the value will be changed to the new value and code will be returned zero. Otherwise, code will be either error_table_$no_property or error_table_$match_fail; if the property exists at all its value will be returned.

HCS_$LIST_PROP

This entry obtains the whole property list for a segment.

```
dcl hcs_$list_prop entry (ptr, ptr, fixed bin, fixed bin,
    fixed bin(35));

call hcs_$list_prop (p, storage, max, actual, code);
```

The pointer "storage" points to a block of space declared like this:

```
dcl 1 prop_list_retarray based (prop_list_retarrayp) aligned,
    2 room fixed bin(35),
    2 used fixed bin(35),
    2 data (1 refer (prop_list_retarray.used)) fixed bin;
```

where "room" is input and "used" is output. In this space, property structures are packed, according to the following declaration:

```
dcl 1 property_retvalue based (property_retvaluep) aligned,
```

```
    2 name char(32),
    2 vl fixed bin,
    2 value char(0 refer (property_retvalue.vl)),
    2 nextword fixed bin;
```

If not enough room is provided, an error code is returned along with a partial property list.

MODIFICATIONS TO STATUS

If a segment has properties, hcs_$status_long and hcs_$status_for_backup return a bit (1) flag which indicates that the property list is nonempty, for the convenience of the hierarchy dumper and the dir_info subsystem.

Modifications_to_Existing_System_Software

Not many changes to the current system are necessary, since the property list is primarily designed for user applications.

COPY

The copy command copies property lists when "-a" is specified.

BACKUP AND RELOAD

The hierarchy dumper and reloader dump and reload property lists.

SALVAGER

When a directory is rebuilt, the property list is rebuilt. Threading errors are repaired and unthreaded entries recovered in the same way that names are rescued.

COMP_DIR_INFO, SAVE_DIR_INFO

These tools save and check the property list.

New_Commands

No new commands are strictly required by this facility, since the applications will provide whatever property manipulation functions they need. However, a few system commands may be useful for cases where one is setting up an application.

ADD_PROPERTY

This command adds a named property to a segment or directory. The star convention works.

GET_PROPERTY

The get_property command/active function obtains the value of a single property or the whole the property list.

get_property path -all

get_property path property_name

The star convention is accepted.

DELETE_PROPERTY

This command deletes a named property from a segment or directory. Star convention works. The control argument "-all" deletes all properties.

APPLICATIONS

A few possible uses of the property list are described below.

Subsystem_Use

The most important use of the property list is in extending the Multics storage system directory so that catalogue data about segments can be kept associated with the segment. The ability to do this frees the subsystem which needs one small data item maintained from constructing a "parallel directory" with the attendant problems of access control simulation, salvage, etc.

CONSISTENT SYSTEM

The Cambridge Project's Consistent System wishes to record a "data type" for each segment catalogued by the system, so that the data-manipulation operations defined by the system may find the internal representation of data segments in the Multics storage system. Currently this is accomplished by maintaining a table with one entry for each branch in the directory, with attendant overhead. By using the property list, the "consistent_system" property could be used to store this information. The Consistent System is also a system which unifies many statistical and data-organizing subsystems. If any of these

subsystems need a catalogue item maintained, the property list can once again be used, without conflicting with other applications. An informal registry of property names may turn out to be a useful idea in the long run -- for a start, naming the property after the subsystem should be adequate.

Small_Segments

The property list provides the solution to a problem which arises whenever an application needs to deal with small amounts of data which should logically be a separate segment in order to take advantage of the Multics protection mechanism. Application designers are often reluctant to waste 1024 words of storage to store about ten words of actual data.

Ring_1_Segments

The current confusing mess of delete commands, rename commands, etc., one for each different ring-1 subsystem, could be cleanly fixed by use of the property list. Ring 1 data bases would obtain a property which told what subsystem managed them, e.g. "mbx". Users would then use the regular delete, rename, and so forth: these programs would be on the lookout for an error code indicating that the segment was an inner-ring segment. New entries into ring 1 would be provided fo delete, rename, etc., and these entries would first get the property, and then dispatch to the appropriate subsystem.

This proposal could be implemented without the property list, by perpetuating the current crock of coding the subsystem into a name suffix. The same feature could be provided by adding an attribute of all segments, which is used only in ring 1. But again, the property list provides an elegant and compact method of adding another feature to the system.

Courtesy_Lock

The conditional-set primitive can be used to implement a new convention, the "courtesy lock convention." Commands which honor this convention would set the "Multics.courtesy_lock" property to the lock_id of the process. If edm, qedx, archive, etc. all respected this property, then the possibility of two users editing the same segment at once could be minimized. The editors could print a warning if the property could not be set, and continue anyhow (since often the editors are used for interactive search), perhaps with the "w" request disabled. Higher-level process-coordination schemes, including analogues for notification when a lock is unlocked, can obviously be proposed once the property list is in place: the "Multics.courtesy_wake" property

could be set to a list of mailboxes to send a message when unlocking.

Use_by_I/O_System

Currently the PL/I input-output system cannot reliably determine whether a multisegment file is of sequential or indexed organization. The "Multics.attach_description" property could be standardized to contain an attach description adequate for attaching the file represented by a branch. Thus, all the attributes necessary for a file become attributes of the file. We would then be able to implement Bernie Greenberg's proposal for an attribute merging scheme similar to that used by OS/360, and be much closer to support of traditional commercial data processing. These changes would simplify application programming by providing a higher degree of device independence. Since the branch merely represents a file, programs could reference zero-length segments which carried attach description properties referencing tape_ansi_ or tape_ibm_, transparent to the application.

Use_by_msa_manager_

The multi-segment area management programs used in the library maintenance software were concealing segment numbers within the bit count of the area segments for a while. This problem was solved by other conventions, but the property list would have been the natural way to provide storage for a small data item which could not be placed in the segment itself, but which had to be associated with the segment.

Use_by_Profile_Manager

When abbrev rehashed the profile, the date for check_info_segs used to get destroyed. If the date were kept in the "Multics.cis_date" property of the profile, then check_info_segs and abbrev would not have to know of each others' existence. Furthermore, new applications which need a per-user data base could be written and debugged without the temptation to create yet another per-user segment.

GCOS_Simulator

The GCOS simulator and GTSS simulator need someplace to store file attributes, such as password, blocking factor, and GCOS permissions, which can't go in the file and shouldn't be separated from it.