To: Distribution

From: A. Kobziar

Date: 10/10/75

Subject: New Storage System: Data Recovery (Part 1 - Salvaging)

Introduction

The following discussion introduces the framework for the new storage system data recovery design. Specifications for the new storage system were given in MTB-110. This bulletin is concerned with the part of the data recovery task currently termed "salvaging." (The remaining part is backup and retrieval.) Data recovery mechanisms exist because of imperfections, both in hardware and software. The reason for a salvager redesign is to increase two important Multics attributes, availability and reliability. Availability implies that stored data should always be dynamically accessible at the demand of any user, while reliability implies no loss of stored data as well as the safe storage of the security information used to protect the stored data.

This MTB proposes a major change to today's salvaging operation. To accommodate storage growth, salvaging will become dynamic and distributed. More of the errors corrected by the salvager will become user visible. An implementation plan which chronicalizes the design decisions still to be made is given in the companion MTB-221 "New Storage System Salvager Implementation." In order to explain why this MTB's design is being proposed, some background material is presented first.

Storage System Overview

As a first order approximation, the Multics storage system can be viewed as a logical organization for an array of file maps. This logical organization can be broken into two parts: directory control and storage control. Directory control handles the logical structuring of the user data and stores the security information. A directory consists of objects (branches, names, acls, etc.) whose data is held in structures. Relations among the objects are implemented by threading the structures together. Storage control manages the file map arrays. A file map is

nothing more than an array of physically sequenced keys to the stored data.

Stored data can be "found" by only one method: logically traversing through a hierarchial structure of directories. Since directories have internal structure, a successful logical traverse requires a physically correct internal structure. Errors have been caused by human, probabilistic (hardware), and even cosmic (unknown, such as lightning causing a power outage) actions. Because only the human cause of these errors can (theoretically) be eliminated, error detection and recovery are necessary. The mechanism used for this purpose currently is the offline salvager. The system is crashed upon detection of an error and correction is achieved by running the salvager (thus making the system unavailable for useful work). Such operation is necessary today, but its cost is too high for the service provided, since most of the directories salvaged have no errors.

New Storage System Structure

The new Storage System (NSS) splits current directory branches into two parts, the logical attributes and the physical attributes. The physical attributes are stored in a self-consistent format, the volume table of contents entry (vtoce) which contains a uid pathname. The connection between a NSS directory branch and a vtoce is logical in both directions. This design is inherently more costly to process for the salvager as well as for the storage system because two disk references, one for the branch, the other for the vtoce, are often necessary where one sufficed previously.

Projecting the present salvager's operation into a NSS format gives running time estimates of 6 hours for a 100 disk drive system. Performing the same operation with a multi-process salvager could cut this time down by 1/2 to 1/10 depending on the hardware configuration. Unfortunately, the near future capacity doubling of the M300400s makes even the multi-process approach unacceptable.

Current directory control is coded with the assumption that threads and relative pointers are always valid. Thus a brief description of the salvager's action would be that its primary purpose is only to prevent faults on thread and relative pointer references. A walkthrough of the salvager code reveals that directory control relies on few of the other parts of a directory object's structure. Other benefits derived from salvaging are garbage collection (directory compaction and the freeing of space used by process directories and hardcore segments), and quota verification. Some cross-checking on acl structures and access-class relationships is done in an attempt to establish security non-compromise.

The salvager also checks for reused addresses by recording all page assignments in a new free storage map which replaces the old one at the end of salvaging. This task has been split out of the directory salvager by the VSS design, since every volume (disk pack) now contains its own map. A salvage operation over a volume will be performed by a volume salvager, to be described later.

Terminology

Before proceeding any further, definitions must be given for the terms used. The term "salvaging" is misleading in its innate description of the current code's function, since an operational description is mostly "directory checking" with correction occurring infrequently. When "salvager" is used it it will refer to today's operation. For the proposed design, compound terms will be used to more clearly indicate which operations are being discussed.

"Directory checking" is defined as that code which detects errors in directories. "Directory salvager" defines that code which corrects and compacts directories. "Connection checking" refers to that code which checks branch-vtoce connections. "Volume salvager" refers to that code which performs garbage collection on a volume.

Specifications for the Directory Salvager

The directory salvager is first viewed as a black box with input, output, and environmental specifications. The following describes the input and output constraints:

1. The input is a bit string and some (read only) context predicates.

2. The output is a valid directory (this includes a null directory).

3. Given a valid directory as input, the output is the same valid directory. This can be called the non-destruction rule.

    a. Optionally, given a valid directory and a length as input, the output is a valid directory which includes the input valid directory as a subset. Valid objects that exist within the input length will be connected to the appropriate places. This is the reclaimation option.

4. If an invalid directory object is found, then it will be changed to be valid only if no security compromise can occur. If it is changed then a possible loss of correctness

will be indicated. Otherwise, it will be discarded. This is the object acceptance criterion and the inverse is the discard criterion.

A few words about the use of the words "validity" and "correctness" are in order. An object is correct if its data has not changed by any means other than by user calls to storage system entry points that are provided specifically to change the data. (Correct data is simply data that has not been clobbered by the system.) Only certain correctness losses are detectable.

An object is valid if its structure conforms to the rules that are implicitly given by the storage system implementation. A minimum set of validity rules is defined by a particular implementation. The directory salvager can, of course, validate all possible structural parts, guaranteeing validity irrespective of any storage implementation changes. This extra checking guarantees that all errors (clobberings) which span both data and structure will, if possible, be detected. Thus the probability of detecting correctness losses is increased.

Design Basics

Clearly, it is only necessary to rebuild directories that have errors in them. All other rebuilding is wasteful and adds to the cost of the service. If the goal of service continuity is put aside for the moment, it would be acceptable to rebuild only the one direectory that caused a crash. In the current salvager, consideration of reliability and garbage collection had made us willing to spend the processing time required to salvage all directories, in the belief that other inconsistencies might exist and would cause crashes shortly into the next bootload. We cannot afford such action on larger hierarchies because salvaging time increases linearly with the size of the hierarchy.

A deeper look into the use of the current salvager at external sites reveals another purpose, that of restoring the confidence level in an "intact" hierarchy back to 100%. (Here "intact" is used to convey the ideas of correctness and validity.)

It is proposed that directory structure checking become an integral feature of the online operation of directory control and that the notion of a separate salvager subsystem be dropped. Error detection will be done dynamically, and corrections will be done by online rebuilding. Dynamic checking can be visualized as the breaking up of the current salvager into two parts, scattering the checking function throughout directory control, and retaining the directory rebuild function. A salvage of the entire hierarchy will still be possible, but will be rarely used.

The economics of dynamic checking indicate that it will be more expensive than today's offline strategy on stable hardware configurations with small disk complements. Part of this cost can

be written off as that necessary for utility operation; i.e. with dynamic checking and online rebuilding, the mean time between failures should increase, and down time minimized to that necessary to repair failures not caused by the storage system. Also the checks are applied in direct proportion to the activity of a directory; a quiescent directory is not checked. A significant cost reduction will be made by altering structures to decrease checking time and to increase error detection probability. Since these costs can only be given in ballpark figures today, part of the design process will be to measure the actual costs on a NSS system before deciding what checking is viable.

Environment

The environment of the directory salvager is considered next. As is true for the offline salvager, the directory salvager relies on correctly functioning lower level machines. Both today's salvager and the NSS directory salvager assume that hardware, page control, and the syserr mechanism work. In addition, the directory salvager will assume that directory locking is also functioning. These assumptions can be made safely as long as errors from lower level machines are either processed in the lower level machine or are random. A random error distribution guarantees that the directory salvager will eventually run during a time period when no errors occur, and therefore will return a valid directory.

As insurance against non-random errors that are not detected by the directory salvager, a small array of invocation times and errors found will be kept in every directory header. If the directory salvager is invoked too frequently, it will inform the operator that a possible loop exists. A review of the errors found should help in determining whether hardware or software is suspect.

The ability for a boot to always get to command level is an important factor in the confidence level in Multics. The offline salvager's contribution here was to guarantee structurally valid libraries. The equivalent confidence in the new Storage System will be achieved as follows: Part of every boot will be to run the volume salvager over the root's physical volume and to directory salvage the root, >system_control_1, and any other important system libraries. The inclusion of salvaging as an integral part of Multics boot relies on a hardcore partition as proposed in MTB-213. If the answering service cannot be started, a reload of the primary system libraries could be performed. Once command level is reached, every site is at liberty to specify more checking in its startup sequence.

Distributed Checking

So far the directory salvager has been viewed as a black box. The following section describes the specific checks and structure changes that are proposed.

At the end of each structure a checksum field and an owner field will be added. The owner field will contain the directory uid for directory threaded objects, and the entry uid for entry threaded objects. A length field and an object type field will be added to the front of each object. Each threaded list will have a unique count of the number of members in the list. Since this is already true for all lists except initial acls, the initial acl total count will be replaced by an array of individual list counts.

Therefore all directory objects will have the following format:

Based on directory structure statistics at MIT, the proposed structure modifications would increase an average directory by 3%.

The following checks can be made by directory control. Each check can be made independently of the others; thus the final installation will have the most viable combination, as determined by cost/performance studies outlined in MTB-221.

1. Change all storage system procedures that calculate relative addresses to check if the address is 0 before using it. In most cases this can be done by adding one instruction when picking up relative pointers. For proper operation, the end of a valid thread will be some value other than 0.

2. All directory objects will have a type id in their structures. All references to a thread pointer or to the item itself, will first check for the correct type value. This check will probably add three instructions to every reference. In a similar manner, the length and owner fields can be checked.

3. For all directory objects, templates are formed which test all the constant bits. For ASCII data this would translate to testing the first two bits of each character to be 0. For directory headers and entries this would translate to testing that all pad fields remained 0. These checks can be

implemented via extended instructions and a test. Template checking will first be used by the directory salvager when rebuilding a directory.

4. Finally, each object will have a checksum stored along with its data. A checksum calculation would take as many instructions as the length of the object plus two more for a comparison and transfer. The cost of storing a checksum when an object is created or changed is negligible if we assume that the number of directory references is much greater than the number of directory modifications; thus it should be calculated along with each modification and used by the directory salvager. Checksums will be calculated for only the relatively constant data fields in a structure, not for items such as date-time-used.

An easily implemented installation option would be to template and/or checksum all access names during an access mode calculation. Access_mode already references exception bits in the pds and thus would require only two or three extra instructions to check anotner exception bit.

ACL Errors

When an acl error occurs, the current directory strategy of sharing access names creates problems in retaining any information from the valid acl entries. All acls that share a bad access name must be deleted as no secure method exists for protecting the integrity of an acl. It is proposed that an acl out-of-service condition be supported by the storage system. Replacement of the acl would be required to turn on service. But the name sharing strategy has often produced many (if not all) invalidated acls in a directory. Thus multiple corrections for one error are required. If acl errors are frequent enough in NSS then sharing of access names should be dropped. This change would also have the beneficial effect of localizing all branch attributes, thus reducing page faults. Sharing within a branch would still be supported.

The cost of not sharing acl names is rather high, with an average increase in directory size of 45%. Even if the savings obtained from reduced page faults (due to the localization of the acl) are included, the sum will still show a cost increase. Recovery of the cost increase could be accomplished by implementing variable size hash tables.

Directory Control Changes

As well as type and owner field checking, certain bounds and cross checks of structure values will be added to directory control when it is in the explicit interest of a procedure to

decrease its gullibility. Several checks of this kind already exist; for example when acls are listed, the number found in the acl list is compared to the count in the branch.

Currently the count of sharers kept with each acl name is used to allow deletion of the name. If the count was incorrect, then reassignment of the name slot would change a person or project name on several acls. For this reason, it is proposed that access names not be freed until a rebuild is performed. A reference to a name with a zero sharing count will be one more form of error detection.

Directory Allocation

A simplification to the directory allocation scheme is proposed. Instead of maintaining several different fixed size free lists, all allocation requests will be placed at the end of a directory. Slots that are freed will be zeroed and not reused. A total of the freed space will be kept so that the necessity of compaction can be determined. This strategy has the desired effect of isolating the introduction of errors. For example, a new branch will have its names and acls physically as well as logically attached. In case of modifications (deletions and additions), not reusing the freed slots allows the detection of cross threading errors, something the current salvager does not check. Thus we are introducing segregation in an attempt to lessen the occurance of errors that spill over (affect more than one branch).

Now that allocation can accept any reasonable size request, links can be stored more compactly. Also the introduction of new objects into a directory need not consider the current block size limitations. Changing sizes of current structures is also facilitated. Implicit in this suggestion is that directory space management will be done inline by directory control because it is so simple. If variable size allocation is adapted, then the proposed new directory structures would only increase an average directory's size by 5% rather than 3%. If variable size hash tables are implemented, then a net size decrease of 30% could be achieved.

Triggering Rebuilds

Whenever the freed space total and the count of directory attribute modifications exceeds some threshold, the directory salvager will be invoked to perform a rebuild. Here we have achieved the desired property that the more a directory is modified, the more often it is validated. It is not necessary to count directory read operations because the new storage system design does not require any directory modifications in order to read (search) directories.

For dynamically detected errors, the mechanism used to trigger the directory salvager is the following: whenever a directory is locked, a handler for the "invalid_directory" condition is set up to call the directory salvager. After the rebuilding, control is transferred to the statement following the lock call, thus repeating the function on the rebuilt directory. Internal directory control procedures need only signal whenever an error is found. All procedures which lock directories must be checked for code which will operate properly when restarted after the lock call - for instance, variables assigned before the lock call cannot be reassigned after the lock call.

Error Reporting

The methods used today for reporting errors detected by the salvager are inadaquate. Of significant concern to users are missing branches, bad names, and lost acls. Although the salvager prints all detected errors, no method exists for distributing these messages or issuing warnings. There is also a problem in deciding who should receive the messages.

Both the directory salvager and the volume salvager will use the syserr mechanism for recording errors, as the syserr log is the permanent record of system events (especially detected failings). The log can be processed online in order to detect error patterns and maybe even predict hardware failures. To reflect errors to users, flags will be set. Bad names and missing branch flags will be set in the directory header while an invalid acl flag will be set in the branch. The current action of deleting invalid acls will be changed to retain the acl for listing purposes. Directory control will treat the invalid acl flag as if the acl was null (acl out-of-service). The invalid acl flag can be turned off by either deleting the entire acl or replacing it. No action for missing names and branches is currently planned, as these are relevant to Multics search rules and could affect every process. For example, if a name was missing in >sss then every process might be stopped until the flag was reset. In the future such errors could become visible by changing the search of such directories to signal some condition. The default action would be to ignore this signal.

Storage Control

Storage control errors are handled by a volume salvager. The input and output specifications for the volume salvager are as follows:

1. The input is the string of bits that comprise a volume.

2. The output is a valid new Storage System format volume.

3. Given a valid storage volume, the same storage volume is returned.

4. If an invalid vtoce is found, it will be deleted. If a reused address is found and the vtoce appears to be a directory, the volume salvager will invoke the directory salvager on this directory. If no errors are found, then the page in question will be awarded to the directory and the vtoce set out-of-service. Turning on service to such a directory must be performed by an administrator.

The volume salvager environment is also similar to the directory salvager's. Each establishes exclusive control over its subjects (in this case a disk pack). Also each relies on correctly functioning lower level mechanisms. For the volume salvager this is disk i/o. In the final implementation, the volume salvager should gain control of the volume via RCP. But for now, a direct path to the disk dim will be used.

Disk Packs

To aid in checking vtoces, their structure will be extended to be similar to that of directory objects. The vtoce checksum will cover only the uid pathname and the access-class, as other vtoce fields change too frequently.

Since only a limited reused address check can be made by page control (the user of vtoces), volume checking would normally occur infrequently. Therefore triggering the volume salvager has to be accomplished artificially. One installation option would be to salvage at the time the disk is logically connected. This might be judged too costly (1 - 3 min. per MSU0400), so that scheduled volume salvaging could be implemented for slack time periods.

As well as salvaging all vtoces, the volume salvager will reconstruct the volume map and will check for reused addresses. A reused address involving a directory will be resolved by asking the directory salvager if any errors were found in salvaging the pages that include the reused address data. A finding of no errors would result in awarding the page to that directory. If errors were found, then the rebuilt version of the directory with a zero page would replace the bad one, and a retrieval request for that directory issued. (A more detailed description of directory retrievals will be given in the backup MTB.) A reused address on a segment would result in a null address award (equivalent to zeroing), an out-of-service indication, and a retrieval request. A second pass over the volume will be made to handle any directories that were the first claimants of reused addresses.

Branch - VTOCE Connection

The new storage system design includes the dynamic checking of the logical connection between the branch and vtoce at activation

time. The resolution of an error at this time should be as follows:

1. Check the vtoce checksum. If it is correct then mark the branch as unconnected. A user encountering an unconnected branch can either delete it or issue a retrieval request for its vtoce. A future addition might be to allow scanning of volumes for an unconnected vtoce entry with the matching uid and upon finding one, connecting the branch to it.

2. If the checksum is wrong, then mark the vtoce out-of-service and issue a retrieval request for that vtoce. The user referencing the branch would receive the out-of-service error immediately and could try again at some later time.

One future extension should be mentioned. Whenever a directory retrieval is performed, instead of replacing the contents in toto, the version from backup and the existing version could be logically coalesced. This would prevent loss of new branches. In any case, notice that although a retrieval can return already deleted branches, the correct action is taken at activation when a connection mismatch is detected.

Loops

In the effort to preserve all possible information, we have chosen not to delete objects but to mark them as having errors, and allowing users to issue retrievals. Unfortunately, there is no guarantee that the retrieved information is correct - in fact it may have the same error. This is a loop which only a user can detect. The resolution is that, if necessary, a previous copy retrieval should be tried, ad infinitum.

An apparent loop also exists in the specification of reused address processing. Assume that the volume salvager detects a reused address when processing a directory vtoce. It asks the directory salvager for some advice. But the directory salvager, in formulating its opinion, can get a reused address signaled from page control, and this would invoke the volume salvager! This sequence is prevented from happening if we insure that all addresses in a particular directory are unique (done by the volume salvager) and that the volume salvager has exclusive control of the disk pack (thus page control cannot signal a reused address on it).

Loose Ends

Purposely saved until the end is the subject of quota validation. The elimination of offline salvaging implicitly dropped this function, since it can only be done on quiescent subtrees. It could be performed online only if there was a guarantee that this was the only process looking at the subtree. One approach to achieve this would be to turn on security out-of-service for all

components in the subtree. Once we assume or take action to provide exclusiveness, a procedure which sets the used values in an aste must be provided. It is proposed that quota validation become a part of the administrative mechanisms used in determining volume usage charges.

While on the subject of charges, notice that the directory control checking design has transformed the collective aggregate cost of offline salvaging into a process assigned "pay as you use" part of the storage system. For physical volumes that are wholly owned by projects, even the use of the volume salvager as the garbage collection device is automatically charged to the correct project.

Summary

1. The salvager is split into three parts: a directory salvager which rebuilds directories, scattered checking in directory control, and a volume salvager which checks for reused addresses and rebuilds the volume map.

2. Detected errors are entered in the syserr log, and users are notified of errors by out-of-service conditions and error bits in the directory header.

3. Directory structures are expanded to be more robust and the directory allocation scheme is changed to take advantage of the directory salvager. The costs for an average directory are as follows:

structure changes -	+8%
variable allocation -	-3%
non-shared access names -	+45%
variable size hash table -	-30%

(end)