MULTICS TECHNICAL BULLETIN MTB- 156

To: Distribution

From: Stan Vestal

Date: 21 January 1975

Subject: Management of System Pool Storage

During the recent design review for the new card input facility, there was a good deal of discussion about the way system storage would be managed for the card pool. Several questions were left unanswered regarding the access a user would have to his card image segments while they resided in the card pool.

The attached design memo describes a set of pool manager subroutines which will provide all the management functions needed to make the pool compatible with the Access Isolation Mechanism. The arguments allow the process calling on the pool manager to define the access to be given to the user. Thus, the card input facility will be able to ensure that the user can only read his data without abusing pool storage. The same pool manager software can be used in other applications to allow the user to share and expand data within his pool directory.

This design memo is presented as an answer to the questions from the design review. The changes to the original design are minor, and thus, no further design reviews are planned.

POOL MANAGEMENT STRATEGY

Pool storage used by a system process on behalf of a user will be managed by a set of subroutines which assume that the caller has access to the system privilege gate. The subroutines and the pool hierarchy are described in the following paragraphs.

POOL HIERARCHY

Pool storage is described relative to the pool root directory. This root directory must exist for all pool functions to be performed; i.e., the root will never be created automatically.

The pool root directory will have an access class equal to the lowest access class to be used in the pool. Normally this will be equal to system low. Under the pool root, there will be separate directories for each access class currently active in the pool. The name of each of these directories will be a unique name derived from the bit string which describes the access class. The entry in convert_authorization_ which performs the bit string transformation will return a null string for the system_low access class. Therefore, the directory under the pool root which is used for system low pool storage will be named "system_low" to avoid over-crowding the pool root with branches.

These directories under the pool root form access class pools. Storage within an access class pool will be further segregated by the personic of the user who required the pool storage. This is done by creating separate directories named "personid". The ACL of the personid directory will be as follows:

variable	Personic.*.*
sma	*.SysDaemon.*
variable	*.*.*

The ACL entries for Personid.*.* and *.*.* above will be parameterized in the pool magagement software in order to allow flexibility in pool usage. The process which manages a pool for a given function will control the access a user will be allowed. For card pool management these ACL entries will be set to "s" thus requiring the user to copy his deck in order to modify it. Other applications may choose to use "sm" for Personid.*.* to allow the user to share his data from the pool. However, the user will not be able to append a branch or link to the directory; this must be done by the system processes which manage the pool storage.

Quota will be allocated to each access class pool at creation time. This quota will be optionally allocated to the personid directories within the access class pool at the discretion of a system process. For example, it may be useful to allow a user to

grow his system pool storage, within limits, for certain applications.

If a pool managing process encounters a record quota overflow while copying data into a personid directory, a condition handler will attempt to move quota to the access class directory using the pool manager add_quota subroutine (this is a privileged operation for any access class above system_low.)

The pool managing process will create segments or multisegment files under the personid directory of the requesting user within the appropriate access class pool. If the personid directory does not exist, it will be created by the pool manager at the time the personid pool is opened (initialized). This is not a privileged operation for a pool managing process and is provided by the pool manager only to isolate the structure of the pool hierarchy from other software.

POOL MANAGER SUBROUTINES

The module "pool_manager_" will perform all operations on the storage pool which require system privilege or complete knowledge of the structure of the pool hierarchy. The root argument in the pool manager subroutines may be either a pathname or one of two keywords: "System_Card_Pool" or "System_Tape_Pool". The keywords are used to isolate the pathnames of two common system pools from various system and user programs. The following entrypoint definitions describe the functions of the pool manager.

ENTRY: pool_manager_$init(root, quota, access, ec);

root	keyword or pathname of the pool root (input)
quota	initial value of quota to assign to the access class directory if it did not already exist (input)
access	access for *.*.* to be used on personid directories in the pool (input)
ec	standard system error code (output)

This entry will ensure that an access class directory exists at the authorization of the caller. If it did not exist, it will be created and the value of "quota" will be moved from the pool root, if possible. If the remaining quota on the pool root is less than "quota", the remainder will be moved. The ACL of the access class directory will be set to "sma *.SysDaemon.*" and "s *.*.*". The initial ACL for directories in the access class directory will be set to the value of "access". Access to the system_privilege_ gate is assumed. The error code will be zero as long as the directory exists on return to the caller.

ENTRY: pool_manager_$add_quota(root, quota, ec);

root — keyword or pathname of the pool root (input)

quota — the number of pages to add to the quota of the access class directory (input)

ec — standard system error code (output)

This entry will attempt to move quota to the access class directory at the level of the calling process. The directory is assumed to be upgraded and thus access to the system_privilege_gate is assumed. This entrypoint is intended to be used by record quota overflow condition handlers to restore a useful number of unused pages in the access class pool.

ENTRY: pool_manager_$clean_pool(root, age, grace_quota, ec);

root — keyword or pathname of the pool root (input)

age — allowed age (in days) of data to remain in the pool (input)

grace_quota — number of unused pages of quota to remain on each access class pool directory if not deleted (input)

ec — standard system error code (output)

This entry will delete all entries in each personid directory under each access class pool directory for which the dtm is older than "age" and will delete all personid directories within the access class pool directory for which the dtem is older than "age". The access class pool will be deleted if the directory is empty after garbage collection. If the access class pool is not deleted, all quota greater than pages used plus grace_quota will be moved back to the pool root. Access to the system_privilege_gate is assumed.

ENTRY: pool_manager_$open_user_pool(root, personid, path, ec);

root	keyword or pathname of the pool root (input)
personid	registered person name of the user for whom the pool is intended (input)
path	pathname of the user's pool directory (output)
ec	standard system error code (output)

This entry will create a personid directory under the access class pool directory at the level of the caller's authorization if it did not exist. It will set the ACL of the personid directory to "s personid.*.*" and will move any quota on this directory back to the access class directory. A pool managing process may then freely append branches to the directory "path".

ENTRY: pool_manager_$close_user_pool(root, personid, quota, access, ec);

root	keyword or pathname of the pool root (input)
personid	registered name of the user and entry name of the personid directory (input)
quota	quota to place on the personid directory on closing. If the value is one (1) set the quota to pages used (input)
access	access to be used for Personid.*.* on the personid directory (input)
ec	standard system error code (output)

This entry will set the quota for the personid directory under the access class directory at the authorization of the calling process. The ACL of the personid directory will be set to the value of "access" for Personid.*.*. The quota value will be determined by the calling process. A value of 0 will allow the user to grow segments in his personid directory while drawing quota from the access class pool. A value of 1 is used to prevent the user from growing segments created by the system. The only privilege required for this entry is modify access to the access class pool directory.

ENTRY: pool_manager_$find_pool(root, ac, personid, path, ec);

root	keyword or pathname of the pool root (input)
ac	access class for the access class pool dir (input)
personic	personic of the personid directory (input)
path	pathname of the pool directory (output)
ec	standard system error code (output)

This entry will return the pathname of a personid pool directory if it exists. The primary purpose of this entry is to remove knowledge of the pool hierarchy structure from user software. No privileged access is required for this entry. The error code will be zero if the pool directory exists and is accessable to the user (with at least "s" effective access) at his current access authorization. The value of ec will be error_table_$moderr or error_table_$no_info for incorrect access and error_table_$noentry for a missing directory if the user has "s" access to the access class pool. The value of path will be correct for the requested personid pool directory if the error code is either 0 or error_table_$noentry.

GENERAL NOTES

For each of the entries which require system privilege access, during privileged operations all IPS signals will be masked and a condition handler will be established for the any_other condition. This will ensure as best we can that the process will always take known action while it is operating in privileged mode. All privileges established after entry will be removed before returning to the caller.