Published: 08/18/66

Identification

Summary of the Backup and Multilevel Storage Management System
Gerald F. Clancy

Purpose

This section summarizes the program structure of those procedures and processes that perform the backup and multilevel functions. Operational goals are presented when they relate to issues affecting Multics performance or user related interfaces. It is suggested that section BG of this manual be read before section BH since many concepts alluded to herein are explained within the basic file system documentation.

Introduction

The backup and multilevel system is that part of the Multics file system which manages the use and provides for the reliability of the secondary storage system. Most efficient use of storage is accomplished by moving segments to physical devices whose access speed is commensurate with the segments' relative rate of activity. High storage reliability is provided by the periodic and asynchronous copying of new data placed on secondary storage onto some detachable and preservable medium.

Multilevel

The functions of the backup and multilevel system can, for the purpose of exposition, be segregated into the set of tasks performed by the multilevel storage management system and those performed by the backup system. Multilevel's domain of activity is the hierarchy of storage devices known to the file system. This hierarchy is constructed by placing the fastest devices at the top with successively slower devices ordered below. Slow, detachable storage remains at the bottom of the hierarchy and by its nature provides the illusion of infinite overall capacity. Segments residing on detachable storage remain known to the basic file system until they are willfully deleted by a legitimate user request. Detached segments must have once resided on-line by the nature of the creation process. Therefore unwillful disassociation of a file from the on-line storage system can only result from device to device motion initiated by multilevel. Multilevel's prime objective is to dynamically distribute segments among available devices such that a firm balance exists between the access speed of a segment's residence and the access demand on the segment by system

users. A well tuned system is one where the most used data exists on the fastest possible device while segments in less demand reside in appropriately slower storage.

Each segment activation attempted by the file system demands that the segment access history be scrutinized by a multilevel procedure. Authorization for the segment to be moved to a faster or slower device is given if its history does not meet the residence criterion of its current device. If a move appears necessary, however, other influencing factors may inhibit its initiation. Consideration must be lent to availability of space on a new device, size of the segment, and transactor parameters reflecting the user's desired access performance.

A reasonable balance between segments and devices can usually be maintained by monitoring segment activity. Each segment's access rate is counted by page control and the resulting value permanently maintained and decayed with time within the file branch. Each time a segment is activated by segment control, a multilevel procedure is invoked. The activity value is then used to compute a residency factor which is compared with the current criterion attached to the present device and a move or don't move decision reached. In this case the move is accomplished dynamically by the basic file system as pages are read in and out of core.

Hence much secondary storage management can be accomplished as part of the user's own processes. However, there are developments which can slowly or instantaneously upset a finely balanced storage system. Many segments which justifiably resided on a faster device sometime in the past may lie dormant for a long period. Therefore, their activity value decreases to an impermissible level. Since this condition arises because of zero activity, the basic file system is powerless to remove them in lieu of more active segments. Eventually, the device itself approaches saturation and its device interface module complains asking for relief. The distress awakens a multilevel process which is prepared to locate, analyze and provide relief to overcrowded devices.

The Device Relief Process

The contents of a distressed device can be thought of as segments stratified by activity value. The following diagram illustrates this.

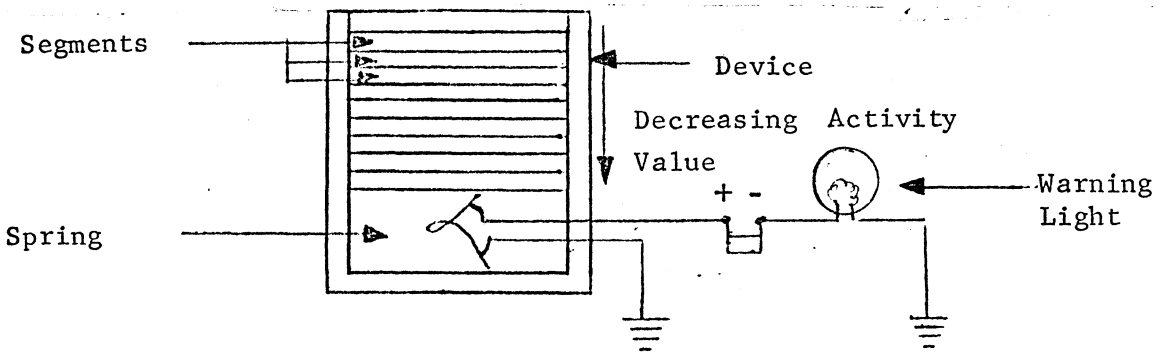

Figure 1.

From this perspective, a storage device is a push-down stack where inactive segments are forced to the "bottom" by new or reactivated segments. Distress arises when tension on the spring surpasses some preset safety value or worse, when the spring will compress no more. Multilevel's problem, then, is to determine how many bottom layers of strata must be removed. Since no dynamic statistics which plot volume vs. segment activity are available, they must be compiled by scanning the file system hierarchy.

If the device has been crippled into uselessness, a guess of the proper removal criteria is made from historical data. All segments with activity less than the guess value are removed to other devices. While thus giving immediate relief, the needed statistics are compiled in case the first criteria was too small. If so, the correct criteria is computed and another pass over the device effected. The guess algorithm need not be exhaustive but may actually be a sampling procedure.

One statistic gathering pass over the hierarchy produces an overall and a per device plot of records occupied versus activity values. Thus, multilevel can isolate the exact source of device overload, begin moving segments to relieve this and immediately calculate the effect this motion will have on other devices. Thus, a second overload of another device might be averted.

Iteration of the above process continues until the overloaded device is completely restored to normal. Whenever one removal pass does not suffice, new material is probably arriving faster than old segments are being removed. Assuming that the removal criteria computation is recent, the stratification obtained is probably still accurate (the strata are in constant motion) or the entire device is filling with very active segments. Recourse is possible by making continuous cycles with a constantly increasing removal criteria, thus preserving the status quo until general system demand relaxes. Once a device is thoroughly

saturated no more segments can be created on that device and it is kept from use until volume is reduced to a reasonable working value.

Each finite secondary storage device is equipped with a "false bottom" so that conditions of saturation may be simulated. The "false bottom" is usually equivalent to the "real bottom" but a mechanism is provided so that this floor may be artificially raised thereby reducing the apparent capacity of a device. If the device is currently operating within safe bounds of its warning thresholds and the bottom is raised artificially, discomfort is thus simulated and multilevel has no choice but to begin removal of resident segments. This procedure is useful should it be necessary to completely evacuate a device prior to a system reconfiguration which is intended to remove the device from service. By raising the floor so that capacity equals zero, multilevel is invoked and evacuation begins. Eventually removal is complete and reconfiguration can occur.

A new (empty) device inserted into the system during reconfiguration is also a cause of some imbalance (but not discomfort) of the type mentioned above. It is felt that this is a vacuous crisis and that the overall storage system will quickly readjust itself regardless of where the new device belongs in the storage hierarchy.

A major assumption in the foregoing discussion has been that space always exists on the intended target device to which a segment is to be moved. This is not always true by the following reasons:

1. A segment to be moved may not fit or might force the new device into discomfort. Movement is not possible in this situation.

2. There may not be a next on-line device in the hierarchy in the move direction.

An examination of a segment activity history on one device provides the move direction and distance up or down the hierarchy. A range of allowable values exists for each device and is updated after the multilevel statistic gathering process. It is not immediately clear what algorithm should be used for this self teaching method but is expected that reasonable values will be reached.

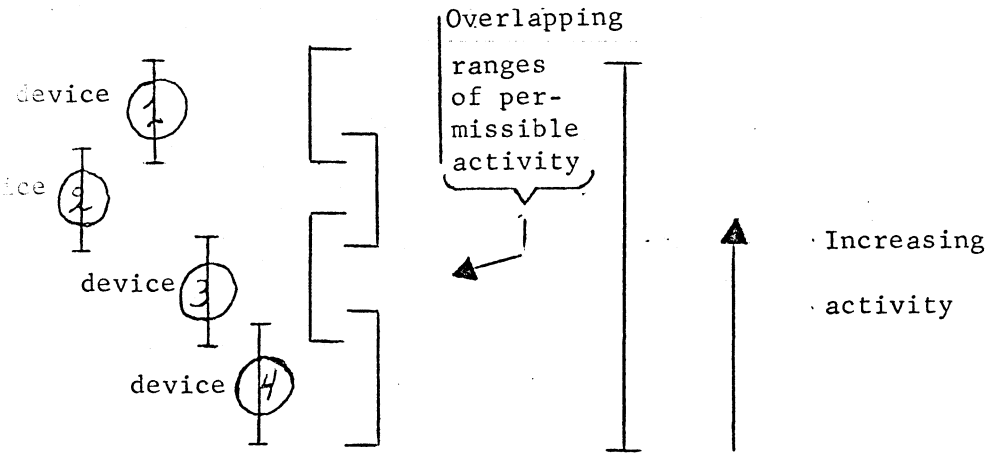

Figure 2.

of overlapping activity ranges is shown in Figure 2.

vice hierarchy naturally terminates with some detachable
e medium which is the slowest device available and,
e it is detachable, has the possibility of having
arge capacity. Hence, detachable storage is always
o receive segment seepage from the on-line devices.
t be emphasized that, although all storage devices
to multilevel and to the user as a continuous hierarchy,
sic file system recognizes only the partial hierarchy
line storage. This discontinuous structure is effectively
by the multilevel software so that no real distinction
e noticed by the user when accessing a segment removed
depths of detachable storage. Figure 3 shows a
tic representation of storage when viewed from the
evel perspective. Squares are storage devices; the
chy spreads downward. Circles represent multilevel
ses which effect segment moves along allowable paths
icated.

loying one of the multilevel processes shown in
3, a segment can be moved from one device to any
. For example, M_3 is the process which moves segments
3 to all other devices as overcrowding occurs.
the process which moves information exclusively
n D_4 (detachable). Motion of this type represents
chanism for retrieving files from detachable storage.
rocess is invoked automatically when a user attempts
erence a detached segment. Normally, however, the
s first told that such a retrieval is required to
his access request. Opportunity is then given
ibit or begin the operation.

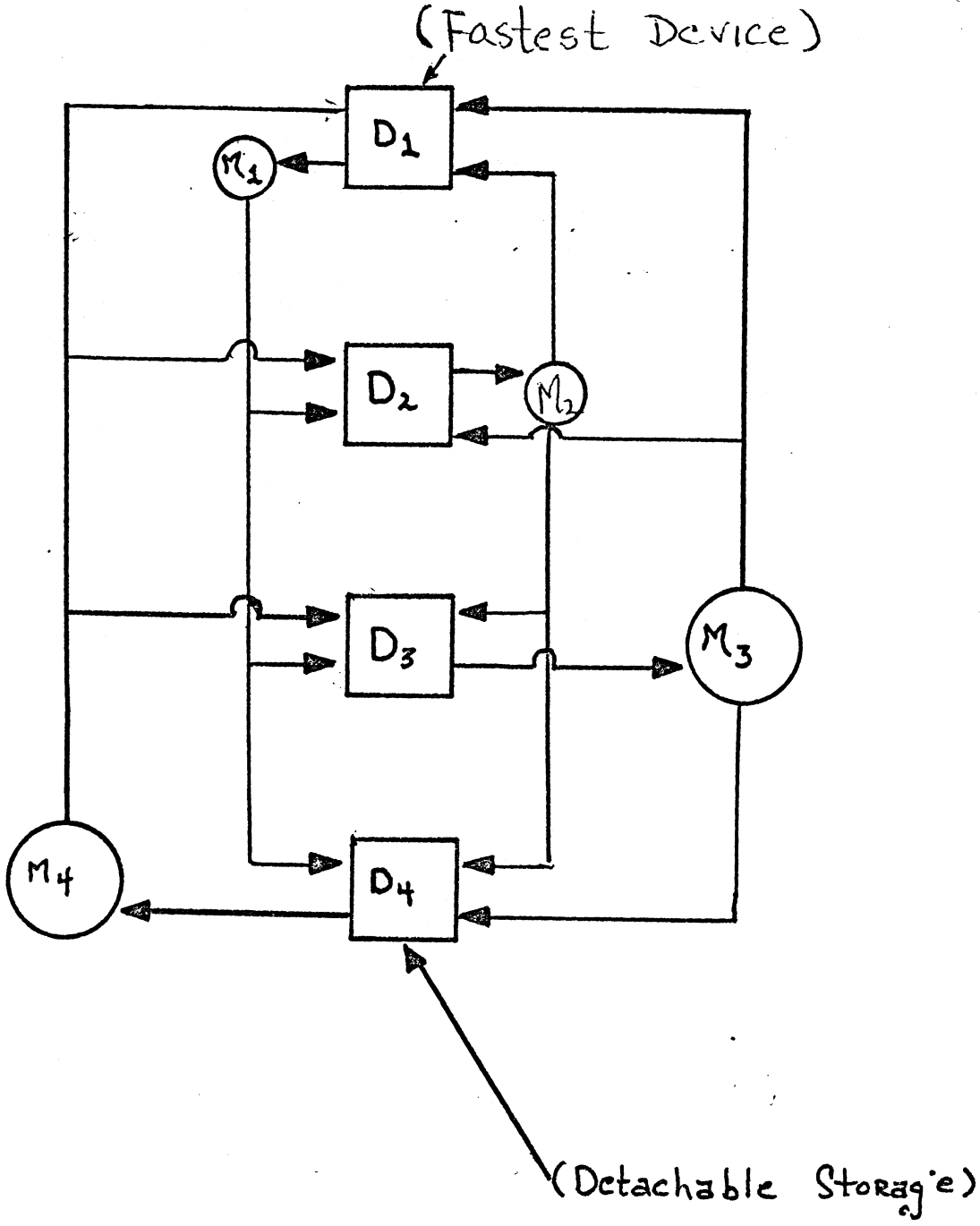

Figure 3

The Backup System

The Multics backup system provides protection against the unwillful destruction of any file known to the file system. Such reliability is maintained through the services of an ever-present daemon whose constant operation is assured. In brief, backup is accomplished by the asynchronous copying of all recently created or modified segments as they are discovered within the file system. Copies are made in duplicate on some form of detachable storage which can be physically removed and externally safe-guarded against pilferage and destruction. This daemon enjoys freedom of the file system hierarchy and is therefore privileged to travel, examine, and read most segments at its own leisure. This standard backup daemon is, however, denied access to those segments deemed highly secure. Backup of this type of information is completed through the services of another daemon which enjoys full security clearances.

Backup copies of segments are unknown and inaccessible to users while those removed to detachable storage by multilevel are completely known by the file system and are accessible to the extent that their reference may cause them to be moved up to on-line storage.

Incremental Dumper

The act of copying file system data on some backup storage medium is known as dumping. Preservation is accurate, complete and guaranteed to last a long time. The retention time of this storage should be sufficiently long to meet the requirements of any individual Multics establishment. The actual dumping mechanism is a set of backup daemons which search out and select for dumping only that data as yet undumped in its current version. Thus, the operation of this process, known as the incremental dumper, produces a continuous stream of output which represents all modification to any part of the on-line storage system. Since modification is forever occurring within an operating Multics, the incremental dumper must be ever watchful and insure sufficient passes over the hierarchy that all data is examined frequently so that backup is current. In practice, the incremental dumper will enjoy an adequate scheduling priority such that one pass will be completed within some time limit. This limit is set externally in order that backup be current to the system requirements.

Each directory entry has an associated date/time last modified (maintained by the basic file system) and a

date/time last dumped (maintained by the backup daemon). Since these dates apply to the entry's associated segment, only a simple comparison is necessary to determine the backup status of any on-line segment. Entries are examined periodically and backup functions performed as required so that total recoverability is both current and complete. The fact that a particular segment has been renamed, deleted, moved or replaced etc., is as worthy of remembrance in backup storage as is modification to actual segment content. Hence individual entries as well as segments are copied by the incremental dumper.

If incremental dumping were carried out continuously from the time Multics first arose and all backup storage generated since were preserved, total recoverability from failure would always be possible by scanning this total volume of information and selecting and reloading only the most current version of all entries and segments.

System Checkpoint Dumper

Obviously, if all incremental storage were preserved indefinitely, all files that existed for some minimum time would be preserved but the total reload procedure becomes prohibitive as the age of the system increases. Instead, certain periodic checkpoint measures executed at convenient intervals are used to consolidate the current total volume of backup storage that must be used to completely reload the on-line portion of the file system. At least three major separate bodies of data seem to be important: a complete set of necessary system files, accounting files, and one complete hierarchy skeleton (copies of all entries of all directories but no other segments). The above items are a minimal requirement which file system storage must have before Multics may operate in any normal fashion (with users). If on-line storage were destroyed, it would be advantageous to restore this information quickly and open the system to general users in spite of the fact that most user segments are not yet loaded.

Periodic checkpoint dumping of all necessary system files, accounting files and the hierarchy skeleton provides recent versions of this data. An optimum period is probably between one or two days and a week. A picture of the processes of incremental and checkpoint dumping is as follows:

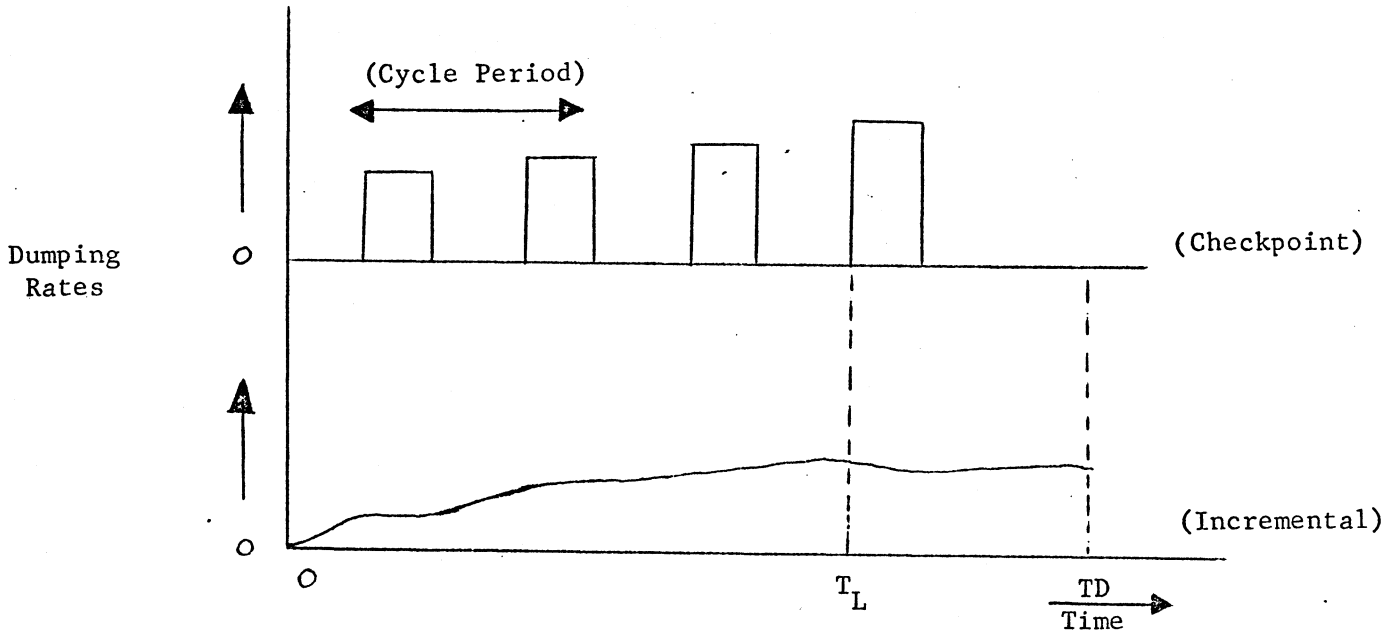

Figure 4.

This process produces a continuous volume of incremental storage and a current latest checkpoint dump. If secondary storage fails at time T_D , the brief reloading of all incremental storage created after time T_L and the checkpoint data which began at T_L will provide sufficient on-line information that the system may operate (in some fashion) with users. After this is done and the system is brought up, remaining segments can be restored by examining the remaining incremental storage (created before T_L) searching for the latest version of each file.

User Checkpoint Dumper

Even though the addition of a system checkpoint dump allows an early return to normal system operation a great deal of effort is still required to scan all selectively dumped files created since time zero.

A further checkpoint dump of all recently used segments is necessary to relieve this burdensome and highly inefficient search. Since the storage volume produced by a user checkpoint is much larger than in a system checkpoint, its period of operation must be correspondingly greater. The complete picture of all dumper output is as follows:

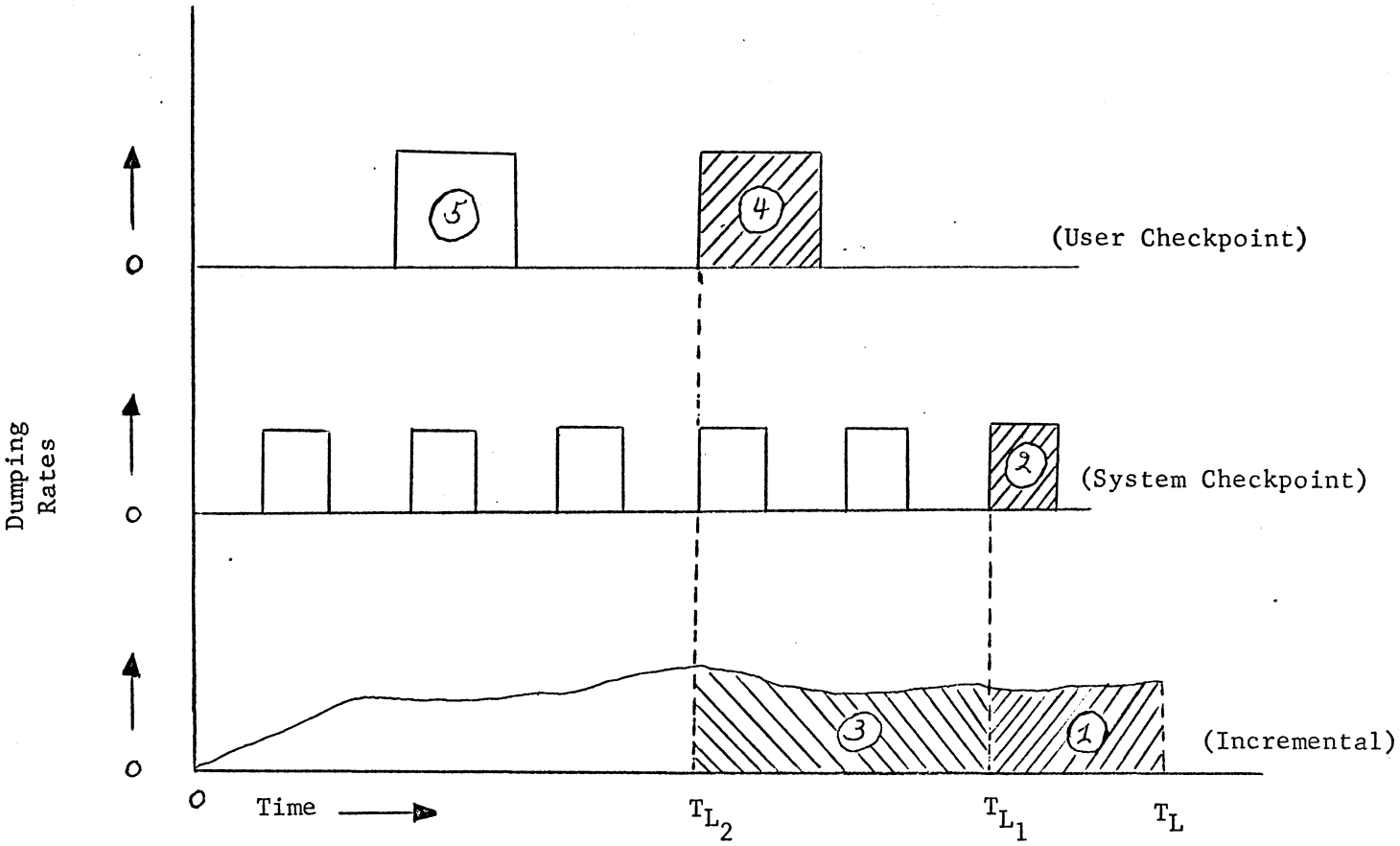

Figure 5.

Reloading

Selective storage contains all hierarchy modifications since the beginning of the last system checkpoint dumper execution. From this portion of incremental storage, only the latest copies of segments are selected for loading. In addition portions of the hierarchy skeleton which are also the latest versions of any directory entries modified are loaded. When this is done, certain portions of the hierarchy exist and will inhibit the replacement of older versions of the same data. The latest system checkpoint storage is loaded next. It contains one complete hierarchy skeleton which can be completely loaded except for those three entries previously loaded from incremental storage. Once the skeleton is processed, it is known exactly which version of every segment must be loaded to completely restore secondary storage. This information was placed in each branch the last time the corresponding segment was last dumped. Therefore, following the skeleton reload, the remaining portion of the system and user checkpoint storage can be searched for exactly those segments which must be loaded.

If on-line storage is lost at time T_L, then complete restoration is achieved by loading the proper segments and directory information contained in hatched areas 1, 2, 3 and 4 above. Since information in 1 updates that in 2, 1 is loaded first. When both 1 and 2 have been processed by the pre-Multics reloader, the necessary system segments are present and normal system operation can begin. Then since 3 updates 4, 3 is loaded followed by 4. Hierarchy configuration is now very nearly restored to its state preceding the catastrophe. Some segments may have been by-passed by the last user checkpoint because of their disuse, but they may be retrieved individually from incremental storage as needed by their owners.

More missing segments could be reloaded by searching the next previous user checkpoint storage (area 5 in Figure 5). This is so since some unused files not dumped by the latest user checkpoint may have been active in the past and would exist on the next latest. However, the time investment required to search such a great deal of information is not commensurate with the relatively small number of files that might be loaded. By avoiding this portion of reloading, it will be noticed that the lowest on-line device in the hierarchy becomes somewhat relieved (compared to its previous state) and therefore a portion of multilevel's function is simulated. The presumption is, of course, that the unused files resided on the lowest on-line device and that a move-off was eminent.

Disposition of Backup Storage

Since the only utility of the checkpoint storage is in anticipation of an on-line storage catastrophe, each time a new user or system checkpoint is created, all earlier versions are of value only if the latest version proves to be unreadable. It therefore seems advisable to always save the last two or three checkpoint dumps of both kinds for such an eventuality. All others created previous to these may be discarded in a manner compatible with security regulations.

Since the complete body of incremental storage contains all segments ever known to the file system and since those segments moved to detachable storage by multilevel are usually quite old then those segments that must be removed already exist within incremental storage. Thus, data within multilevel's detached storage is merely a subset of incremental storage and the need for off-line storage known only to multilevel is eliminated. Whenever a file is to be removed, a simple check is made to insure that

a current copy has been made by the incremental dumper. If so, a truncation is done and no actual output is necessary. If the pathological case occurs and the desired copy does not exist on permanent storage, an output request is issued which will dump the file on any available permanent storage and then truncate.

Some portions of selective storage must be preserved even after it is no longer useful for backup purposes since some segments have been removed from on-line storage and hence are still known to the basic file system but reside wholly off-line on detachable storage. However, all data in selective storage need not be preserved and consolidation can be done if all known segments are preserved by that process.

Consolidation

In order to decrease the amount of incremental backup storage accumulated, a consolidation system is provided and is used by a system administrator whenever the cost of storing voluminous amount of off-line storage is greater than the machine time cost needed to perform the consolidation operation.

Each copy of each segment dumped onto permanent backup storage is provided with an absolute time limit beyond which that copy will not be preserved by the consolidation procedures. Thus, if the limit span is greater than the consolidation execution frequency, then certain copies of segments will eventually disappear from the backup archives. Whenever a segment is removed from on-line to off-line storage by the multilevel removal mechanism a record is kept remainding the consolidation system that although the segment does not exist within secondary storage, it is still known to the basic file system.

The act of consolidation is the copying of certain segment copies from old to newly created backup storage. A given copy will be transcribed in this way only if 1) it is still known to the basic file system or 2) if its expiration date has not yet arrived. For all copies which are still known to the basic file system and copied in this way, the expiration date is rejuvenated by setting it to the current time plus the retention time span.

Dumping Techniques

The file system hierarchy is organized much as a library of segments. As such, some data is naturally dedicated to cataloguing the data segments and, therefore, there are directory segments in addition to the users data segments. Files are created, modified and deleted and these operations are reflected in the directory segments so that the catalogue information is always current. If the backup system maintains an off-line copy of the entire file system, then all segments, user and directory, must be preserved or dumped with sufficient celerity to guarantee a minimum loss of information should on-line storage fail. Directories are of special value since they contain tables of information about other segments. Since a very minor directory alteration would cause a great deal of tedious processing in order to completely backup a directory, a slightly modified viewpoint of the file system hierarchy presents itself. Each user data segment is located uniquely in the tree hierarchy prefixed by a string of successively superior directory entries. Directory entries themselves are distinct entities also preceded by a similar string of entries. Thus attempts are made not to backup entire segments but only data segments and individual entries (pieces of directories). A backup quanta (full logical record in backup storage) consists of a preamble followed by a dumpee. The preamble is a list of successively inferior directory branches which uniquely position the dumpee in the hierarchy. The dumpee is either a full segment or a single entry which must be preserved for some reason. In many cases directory entries themselves and not their associated data segments are modified by users. For example, a renamed or deleted branch or one whose access control list has changed is worthy of backup's remembrance, but dumping of the associated data segment is not required. Therefore, entry modification is detected by the incremental dumper and only a relatively small amount of information is preserved. Only in rare cases are preamble strings followed by an entire directory content (usually when dumping the hierarchy skeleton). A preamble has the property that (1) the n^{th} entry in the string was always extracted from the n^{th} level in the hierarchy and (2) all but the last item in the string is necessarily a directory branch.

Summary

To summarize, six independent multilevel and backup processes have so far been described.

1. A multilevel process which scans the hierarchy seeking to relieve distressed devices.

2. A constantly operating incremental dumper

3. A periodic system checkpoint dumper

4. A periodic user checkpoint dumper

5. A pre-Multics (Phase I) reloader

6. A Multics (Phase II) reloader

In addition, a multilevel procedure lies embedded within the basic file system and is invoked as part of the user's own process whenever a segment is activated. A terse outline of each of the above follows:

Multilevel Move Module

The multilevel move module resides within the basic file system and determines the movement requirements of the segment defined by the directory entry passed as an argument. It is called both by the basic file system each time a segment activate occurs and by the relief process. A segment may be moved up or down within the device hierarchy or left alone. An activity value for the segment is computed from the rate of past access, the transactor priority parameters, size of the segment and the current availability of space on all on-line storage devices. This number is then compared with the allowable limits for the present device and if found incompatible, a new device is specified. If a move seems necessary, indication is given to segment control of which device will give optimum reference performance for the segment's apparent usefulness.

The Multilevel Relief Process

A DIM whose device is becoming saturated, signals the multilevel relief process telling that some preset volume threshold has been reached. The relief process initiates a hierarchy scan whose mission is twofold:

1. Using a past criterion, segments are moved from the distressed device to new homes thus giving a measure of immediate relief.

2. Current statistics relating the usage of all system devices are compiled so that a true picture of current storage and segment activity can be measured.

When the first pass is complete, the effect of the moving just done on overall storage balance is calculated. Measures (another hierarchy pass) are then taken to apply relief to the original device and perhaps adjust other storage to prevent the relief of one device from upsetting another.

This process makes move decisions which may require that information be removed from on-line storage. A record is maintained within the file system hierarchy telling exactly where in off-line storage the removed segments may be found should a future move up (retrieve) be necessary.

Incremental Dumper

The incremental dumper is kept in constant operation whenever the Multics system is functioning. Its sole function is to search out segments and directory entries that have changed since their last dumpment. The entire hierarchy tree is scanned and data destined for dumping is written on some form of detachable backup storage. This storage is saved indefinitely and may be consolidated when economical.

System Checkpoint Dumper

This process is run only periodically and performs a finite task. A predefined set of system and accounting segments and one complete hierarchy skeleton are dumped. The system and accounting files are those files deemed necessary to operate Multics in a normal fashion with users. The skeleton is the set of all directory segments. System checkpoint storage is discarded periodically as more current versions are created.

User Checkpoint Dumper

This process runs periodically (probably with a greater cycle time than that of the system checkpoint dumper). It dumps all segments used or modified since its last running time. A single hierarchy scan is executed with file copies written on backup storage specifically alloted to the user checkpoint dumper. User checkpoint storage is discarded periodically as more current versions are created.

Pre-Multics (Phase I) Reload

The pre-Multics reloader is used to begin restoration of on-line secondary storage following a catastrophe. Its prime objective is to recreate sufficient data within

the file system hierarchy to allow normal Multics operation. The skeleton is constructed in its most current form. When this is done, a complete record of all segments known to the file system and their exact location in backup storage is known. Then reloading necessary system and accounting files allows at least rudimentary system operation. The above data is collected by searching all incremental storage created since the beginning of the last system checkpoint for all latest modifications to the hierarchy skeleton. Next, the copy of the skeleton itself is loaded from the checkpoint such that all directories not already present are loaded. The system checkpoint accounting and system segments are then loaded since the exact copies sought are known. Normal Multics operations can now begin.

Multics (Phase II) Reload

The Multics reloader runs during normal system operation and follows the Phase I reload. It reloads the latest copies of all files found on incremental storage created before the beginning of the last system checkpoint and after the beginning of the last user checkpoint. Since all sought files and their locations are known, the above portions of backup storage are searched once and all required files are placed on secondary storage. During Phase II, user retrieval of segments not present is inhibited.

More

Later continuations of this section will discuss:

1. The I/O subsystem
2. Parallel processing of functions
3. The hierarchy scan mechanism