Published: 12/23/66

Identification

The Hierarchy Reconstruction Process
Gerald F. Clancy

Purpose

The hierarchy reconstruction process is used to begin restoration of the file system directory hierarchy following a catastrophe which destroyed any contents of the secondary storage system. A complete secondary storage reload is executed in two distinct phases: 1) the restoration of all directory segments present when the catastrophe occurred along with sufficient system and accounting segments necessary for normal Multics operation and 2) the restoration of user owned segments. The hierarchy reconstruction process executes before Multics can operate in a normal fashion and performs the tasks of the first phase. The second phase tasks are accomplished by the secondary storage reload process (BH.3.02) which operates during normal Multics operation and restores to the file system hierarchy the bulk of user owned segments.

Introduction

As stated above, the objectives of the hierarchy reconstruction process are:

1. Restore to the directory hierarchy the most recently dumped version of each entry of each directory present on-line when the storage catastrophe occurred. This set of data comprises the hierarchy skeleton.

2. Restore to the file system hierarchy the last dumped version of each segment contained in a predefined set of system and accounting segments.

Once the above steps are accomplished, the hierarchy skeleton exists in the last state before the catastrophe for which it is accurately removable and hence no part of it need be modified by future reloading activities. Sufficient data and procedures are present so that Multics may function with users.

Missing user segments are flagged so that any reference to them will automatically inform their user that restoration is not yet complete. Thus, the system may now be opened to users and the secondary storage reload process (see above) can begin its work.

The latest system checkpoint dump (see section BH.2.00) provides the bulk of detachable storage from which the reconstruction process reloads the hierarchy skeleton and the system and accounting segments. This dump contains exactly this data and was accurate at the time the dump began. All subsequent modifications to the skeleton and the system and accounting segments are contained in detachable storage produced by the incremental dumper after the time when the last system checkpoint began. Thus it is possible to accomplish the first phase reloading objectives by scanning these bodies of detachable storage and restoring to the hierarchy the latest version of every directory entry and non-directory segment found therein.

The method by which the correct set of detachable storage units is identified and remembered is explained in Section BH.4.02. In general, the current reloading unit list is written at the beginning of each newly created detachable storage unit by the output process (BH.4.01) and read by the input process (BH.4.02).

Reloading

The hierarchy reconstruction process scans its assigned units of detachable storage and loads 1) all directory entries which are later versions of other entries already in the hierarchy or which do not yet exist and 2) all segments defined by previously loaded branchs and which are later versions of an existing segment. Due to the nature of the dumping processes, copies of newly vacated entries are dumped signifying the deletion of some hierarchy subtree. When an entry of this type is encountered and supersedes the current entry in the hierarchy, any sub-tree inferior to the entry is also superseded and hence is deleted. Thus the hierarchy skeleton is recreated by the reloading and superseding of entries and segments read from detachable storage until all incremental storage created since the last system checkpoint dump began and the last system checkpoint dump itself have been processed.

The reloading operation is centered about each independent logical record found on the designated portions of detachable storage. Each record is composed of a mandatory header and preamble followed by an optional segment copy (see section BH.4.03 for a detailed discussion of this format). An input request is issued to the input process (BH.4.02) which returns the heading information of an available logical record. The caller is assured that the preamble and segment portion (if any) will be available on a subsequent

call. The preamble is requested next and dissected into its component (and successively inferior) entries. Each entry is loaded into the proper place in the directory hierarchy unless it is superseded by a later version of the same data loaded previously. If a non-directory segment copy accompanies the preamble, a similar interrogation is made and the segment is loaded if required. If a directory segment follows the preamble then it is decomposed entry by entry in a manner similar to the dissection of the preamble. Each entry is then loaded individually if required.

The Modules

The modular constituents of the reconstruction process are process control, preamble processor, segment load module, directory load module and the entry setup module. The following is a brief introduction to each.

Process control - The process control module directs the logical record processing of all records on the assigned portions of detachable storage in the following order. First, it issues an input request and thus establishes a record header. It then calls the preamble processor module to dispose of the ensuing preamble. A return from that module implies that each component entry of the preamble has been loaded if so required. Finally the directory load module or segment load module is called, if necessary, to load the remaining portion of the logical record.

Preamble processor - This module reads the preamble portion of the logical record currently being held by the input process. The preamble consists of a string of successively inferior directory entries. Each is extracted in turn and passed to the entry setup module which will update the entry into the hierarchy if required. The only function of the preamble processor is to present every preamble component to the entry setup module. Once the list is exhausted, control is returned to the caller. The entry setup module may direct the preamble process to abort further processing of the current preamble if it finds that all remaining entries in the list are out of date.

Entry Setup Module - Whenever a candidate entry for reloading is to be considered, the entry setup module is called either by the preamble processor or the directory load module. Associated with each entry copy accompanying the call are 1) a count of the number of entries of this type within its directory current at the time when it was dumped and 2) the date and time

when that count last changed. The number of entries of the current type is established in the directory containing the entry as the size stated in the entry data under consideration if that number is later than the existing count in the directory. Once the size of this directory portion is set, the entry itself is loaded. If a version exists in the directory corresponding to the copy derived from the input medium then it is determined which of the two is the later version and that one is left standing or loaded as required. The above operation may reduce the size of the directory if the new count is later than and less than the old count. In this way, proper directory sizes are established.

Directory load module - Whenever this procedure is called, the process has available a directory segment on detachable storage awaiting input and loading. The directory load module dissects this mass of data into the individual entries belonging to a common directory. For each so isolated, the entry setup module is called to insure that the proper version gets (or remains) loaded into the hierarchy.

Segment load module - The procedure expects that a non-directory segment exists next on the input medium. It reads the required amount of data via the input process and places it in the appropriate location within the file system hierarchy.

Figure one shows the block configuration of the reconstruction process modules.

The Process Control Module

The process control module serves as the main logic program for the reconstruction process. Whenever the process begins execution, control passes to this procedure via the following call.

```
call reconstructor;
```

Each logical record is composed of header information and a preamble record optionally followed by a segment. An input request is issued to the input process which returns the header data of an available logical record. The following information is included in that block.

1. The specific location of the record on detachable storage.

2. Unique identification of the branch entry defining the segment portion if one is included in the logical record.

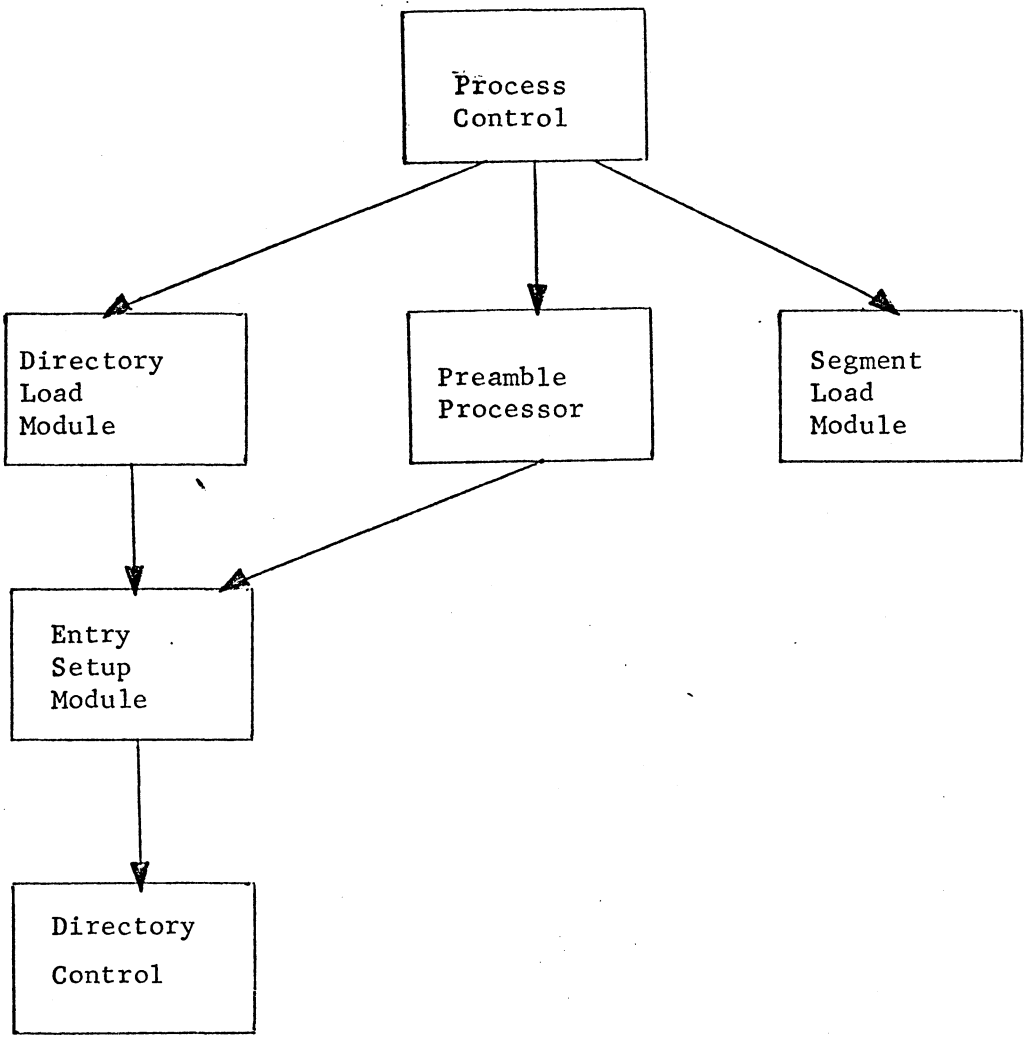

Figure One - The Reconstruction Process

3. The date/time-last-modified of the segment portion if one is included in the logical record.

4. File-removed-switch indicating whether (ON) or not (OFF) the trailing segment was dumped because of a multilevel move-off operation. (N.B. this condition is rare since almost all segments are backed-up before any mo e off operation is required.)

5. File-follows-switch indicating whether (ON) or not (OFF) a segment portion is included as part of the logical record.

6. The type of backup storage (incremental or checkpoint) from which the record is being read.

7. The number of distinct entry components of the preamble.

The process control module then calls the preamble processor to read and load the preamble record which follows the header record. The preamble processor is supplied with the number of entry components in the preamble as an argument.

The preamble processor returns three parameters: 1) the date-time-last-modified and, 2) unique identification of the hierarchy entry corresponding to the terminal preamble component (this entry will be the later of the input entry and the current hierarchy entry) and 3) the segment-already-loaded-switch which indicates, if ON, that the terminal entry defines a non-directory segment and that the segment defined by that branch was loaded sometime in the past and that the segment is not to be superseded by another version, if one exists, within the current logical record.

Process control then calls the directory or segment load module (which ever is appropriate) to process the final portion of the logical record unless one or more of the following conditions exist.

1. There is no segment included in the logical record (file-follows-switch is OFF).

2. The file-removed-switch is ON.

3. The segment version which is defined by the terminal entry of the preamble is not the exact version of the one that constitutes a part of this logical record and hence no loading is required. (The unique identification and date/time-last-modified found in the header record do not match the same pair of parameters returned by the preamble processor.)

4. The segment-already-loaded switch is ON.

5. The preamble string was purposely aborted before all the entries were processed. This means that the segment should definitely not be loaded since its defining branch was superseded. (This condition is signalled by returning the date/time-last-modified and unique identification values as zero.)

Whenever a segment is dumped by the incremental dumper, the dumped copy of the branch defining that segment is incorrect because the retrieval arguments, which are supposed to locate where in the detachable storage archives the segment copy can be found, point to the previously dumped version of the segment (presumably with different contents). Therefore, whenever the reconstruction process loads a segment from incremental storage, the retrieval arguments in the hierarchy must be altered to point to the segment copy just loaded. Similarly, if a segment load from incremental storage is inhibited because of an ON file-removed-switch a similar alteration must take plase since the current retrieval arguments are incorrect.

The last act of process control is to insure that the retrieval-trap-switch in those branches defining non-directory segments is set to signal the presence of the segment. (OFF if the segment is loaded, ON otherwise.) For this purpose the segment-already-loaded-switch is interrogated. It was set either by the segment load module if a segment was just loaded or by the preamble processor if a segment version exists in the hierarchy and was not superseded. The retrieval-trap-switch is set ON only if the segment-already-loaded switch is OFF.

The Preamble Processor Module

The preamble processor reads one complete preamble from detachable storage, dissects that data into its component entries and for each entry, calls the entry setup module so that hierarchy reloading may take place. Each call to the setup module returns the unique identification and date/time-last-modified of either the entry from the preamble processed by the call or its corresponding entry already in the hierarchy. The choice is determined by which of the two entry versions later and hence left standing in the hierarchy. If both parameters are zero on the return from the call then the setup module has signaled the caller that further processing of the preamble should be aborted. When the last entry in the preamble has been processed, return is made to process control with the date/time modified and unique identification last returned by the setup module.

The preamble processor is called by the following

```
call preamble_processor (dtm, uid, n, sal);
```

In this call n is the number of component entries to be found in the preamble; dtm and uid the returned date/time-last-modified and unique identification of the entry ultimately left standing in the hierarchy and which corresponds to the nth preamble component. sal is the segment-already-loaded switch returned to the caller.

The PL/I declaration of the parameters used in the call is:

```
dcl dtm bit (72),
    uid bit (70),
    n fixed bin (17),
    sal bit (1);
```

The Entry Setup Module

The entry setup module is called with one complete entry version. Its function is to locate another version of the same entry which might already exist within the file system hierarchy and to ascertain which is the later version. If the current hierarchy entry is more current no action is required. Otherwise the new must supersede the old. If no corresponding entry exists, then the input entry is loaded.

For the purposes of the reconstruction process, a directory segment can be envisioned as being composed of the following bodies of information (a detailed account of the directory structure can be found in section BG.7.00).

1. A set of branch entries which point to other segments.

2. A count of the number of branches (bcount). The only restriction on this count is that is be greater than or equal to zero.

3. The time when bcount last changed (btime). btime must be a legitimate date and time except during the period beginning when the directory was created and ending with the insertion of the first branch. During that period btime must be zero.

4. A set of link entries which point to other links or branch entries.

5. A count of the number of links (lcount). lcount is restricted as is bcount above.

6. The time when lcount last changed (ltime). ltime is restricted as is btime above.

7. An optional common access control list (CACL).

8. A count (0 or 1) of the CACL's, (ccount).

9. The time when ccount last changed (ctime). ctime is restricted as is btime.

Each branch in a directory defines another segment in the hierarchy and therefore contains data defining the location of the segment on secondary storage and the location of the current backup copy on detachable storage created by the incremental dumper (i.e. the retrieval arguments).

Each link and CACL has an associated time indicating when any information within the entry was last modified. Each branch has a date/time-last-modified for any data inferior to the branch. Given two entries of the same type both of which allegedly occupy the same position in the same directory then a simple comparison of the date/times-entries-modified determines the later entry.

Each time the entry setup module is called, the following information is supplied:

1. A directory entry copy read from detachable storage.

2. A count, accurate at the time the entry was dumped, of the number of entries of the current type in the directory (entry types are link, branch or common access control list -(CACL)) (ecount). This count helps establish the proper directory size.

3. The date/time when the above count was correct (etime).

4. The tree name of the directory of which the entry is a member. (dir)

5. The slot number of the entry. (slot) slot is $\leq$ ecount.

The getentry primitive of directory control is used to ascertain if another version of this same entry now exists within the directory.

The getentry primitive of directory control is also used to interrogate the current state of the entry determined

by slot in the directory dir. Return from this call results in one of three situations.

1. The returned type count time (etime) is zero indicating that no entries of the current type have ever been created in the directory specified by dir.

   In this case the entry and the directory (if need be) are created via a call to the putentry primitive of directory control. The directory type size (ecount) and the count time (etime) are set to whatever is specified in the incoming data. If the entry specifies a vacant branch then the date/time-last-modified and unique identification returned to the preamble processor are both set to zero. This signals that there is now no segment defined by the branch and no version need be loaded. Otherwise, the date/time-last-modified and unique identification of the newly loaded entry are returned.

2. Some entries of this type exist but the one specified by slot does not. The present type count-time (etime) is other than zero.

   In this case, the proper type size for the directory must be determined and established if necessary. The proper size is set from a comparison of the candidate etime with that extracted from the hierarchy via getentry. If the candidate time is greater (later) than the existing value then the incoming etime and ecount replace the old values in the hierarchy. If the new ecount is less than the old, some entries must be deleted from the directory along with any segments inferior to the deleted entries. If the new ecount is greater than the old, skeleton entries are inserted to expand this portion of the directory to its proper size.

   It is possible, if the existing ecount is later that the current entry cannot be loaded because it belongs in a region of the directory which is outside of the proper directory size. If so, the date/time-last-modified and unique identification arguments are returned to the caller as zero. If the entry is indeed loaded, then the entry's vacancy switch is tested and measures taken as in case 1 above.

3. Another version of the entry specified by dir and slot currently exist within the directory.

   If the resident entry is definitely later (its date/time-last-modified (dtm) is greater than that of the incoming candidate entry), or the dtm's are equal but the resident

entry was dumped later than the candidate entry then no action need be taken. The date/time-modified and unique identification returned to the preamble processor assumes the values of the same parameters of the resident entry unless that entry defines a vacant branch; then both are returned as zero.

If the candidate entry is a later version of the resident entry then the former must replace the latter. In the case where both are links or CACL's the newer immediately replaces the older. If branches, then the old branch and its entire sub-tree are deleted before the candidate entry is placed in the directory. Control then returns to the preamble processor with the proper values set for the returned date/time modified and unique identification arguments (i.e. those of the entry just loaded).

The entry setup module is called by;

```
call entry_setup (dtm, uid, dir, slot, sal, entryptr,
                  date, count);
```

In this call dtm and uid become the returned date/time modified and unique identification of the entry left in the hierarchy at directory dir and position slot after the setup module has completed its work. sal is the returned-segment-already-loaded-switch. It is set ON only if a segment is attached to the resultant branch defined by dir and slot. entryptr is a pointer to the candidate entry structure, count and date are respectively, the number of entries of the current type in the directory (incoming ecount) at time date (etime).

These parameters are specified as in the following PL/I declaration.

```
dcl (dtm, date) bit (72),
    (dir, slot) char (*).
    uid bit (70),
    sal bit (1),
    entryptr ptr,
    count fixed bin (17);
```

Segment Load Module

The segment load module is called to transfer data from detachable storage to a file system segment by the following:

```
call load_segment (dir, slot, length);
```

Here dir and slot specify the directory and branch of the segment to be loaded. length is the size of the data in 64 word blocks.

The PL/I declaration of the arguments is:

```
dcl (dir, slot) char (*),
    length bit (12);
```

Directory Load Module

The directory load module is called to read a directory segment from the input medium and to construct a hierarchy directory from that data. Whenever this procedure is called, the process has available a directory segment on detachable storage awaiting input and loading. The directory load module dissects this mass of data into the individual entries belonging to a common directory For each so isolated, the entry setup module is called to insure that the proper version gets (or remains) loaded into the hierarchy. It is called by:

```
call load_directory (dir);
```

where dir is the tree name of the directory segment to be assembled and

```
dcl dir  har (*);
```

declares the lone argument.