To: Distribution

From: EJ Wallman

Date: March 12, 1975

Subject: Interim File Recovery: Backup to Disk.

Contents

A. Background.

Several Multics customers and potential customers have expressed concern over the variously estimated times required to recover the hierarchy after its loss. This concern has manifested itself in a requirement in a current Multics contract for an Interim File Recovery feature. The feature is to include two items; Disk Pack Copy (already in implementation as a modification of BOS's SAVE command) and Backup to Disk. The Disk Pack Copy will serve to minimize the time necessary for recovery of valid data from a damaged Disk Pack. The Backup to Disk discussed here will serve to minimize the time necessary for the update of the hierarchy after a RESTOR.

B. General Discussion.

Backup to Disk is an optional feature of the storage system backup facility which must be specifically invoked at a site. It operates in parallel with the normal tape backup. The disk pack prepared by this feature is intended only for reloading and not for retrievals so that there is no requirement for a search algorithm to find specified data items.

Experience on System M at Phoenix has shown that a backup/catchup tape will hold some 2000-2500 records (pages) of data at 800 bpi and will take about 5-8 minutes to process on reload. At 1600 bpi, a tape will hold some 4000-5000 records and will take 10-12 minutes to process on reload. Based on the above and the capacity of a d191 pack, it may been seen that a reload of one [pack_capcity] of data will consume somewhat over an hour of elapsed time. This elapsed time includes the time necessary for label inputting, reel mounting, and, since the reloader uses only a single tape drive while in the ring_1 Initializer environment, tape rewind time.

The BOS TEST command (which uses only one logical channel) takes 12-15 minutes to test an entire area. Allowing for the fact that a reload must perform more functions than TEST, it appears that the time for reloading each 20,000 records of backup data may be cut to 20-30 minutes. This improvement will come from the reduction in operator time for tape handling and not from any inherent characteristics of the disk over those of the tape. The MTS500 and the DSS191 have roughly equal overall data transfer capabilities. A certain advantage will accrue to the disk since the I/O transactions will be up to 4072 words (4 pages minus DCWs) instead of 256 or 1024.

This backup-to-disk feature is intended to be released as a Special only to that site at which we have the committment. Any additional release must be considered on the merit of the Request for Special asking for the release.

This proposal involves the use of the unstructured, removable disk DIM, rdisk_, currently in developement (See MTB-162, "Multics Removable Disk I/O Module). Ordinarily, such an early use of a DIM is unjustified but is viewed as a calculated risk in this case because of the Special nature of the release and the time constraints on the committment. Moreover, backup-to-disk will serve as a test vehicle for rdisk_.

C. Specific Proposal.

1. bk_arg_reader and bk_ss_ will be modified to accept and process a "-disk"/"-nodisk" option pair. The default will be "-nodisk" so as to preserve "normal" operation.

2. Initialization of start_dump, catchup_dump, and reload will be modified to test the -disk option flag in bk_ss_ and to call bk_disk_$init if the -disk option is taken.

   bk_disk_$init will query the user with command_query_ for a Volume ID (Label) and an optional "-init" control argument and will then attach the pack with iox_$attach specifying rdisk_ (and the -write option if dumping). If a non-zero error_code return shows that the drive cannot be attached, bk_disk_$init will notify the user with command_query_ and determine if the attach should be retried (presumably after manual intervention has made the drive available) or if the -disk option flag should be reset, abandoning the disk output for this invocation.

After the pack is successfully mounted, bk_disk_$init will use iox_$open to open the "file" with direct_input for reload or direct_update for the dumps and read the header sector of the pack (iox_$read_record, key value "0") into a 64 word structured area named bk_vol_header (probably most conveniently located in bk_ss_). (NB: The header sector of the pack is initially defined as sector 0 but, if the Resource Control Package requires a standard volume label for <u>all</u> packs, it will have to move to the first free sector following that label.)

64 words is chosen as being the "natural" data increment of the disk. Not all 64 words need be used, but such a selection is consistent with the manner in which rdisk_ treats the pack as a direct file, keyed on sector number. The structure of bk_vol_header is as follows...

```
dcl 1 bk_ss$bk_vol_header,          /* header from pack */
   2 volume_header_id char (128), /* backup pack identifier */
   2 volume_id char (32),         /* pack serial number */
   2 unique_id bit (72),          /* set at initialization */
   2 first_clock fixed bin (71),  /* time of initialization */
   2 last_clock fixed bin (71),   /* time of last header update */
   2 first_rec fixed bin (35),    /* sector no. of 1st data */
   2 next_rec fixed bin (35),     /* sector no. of next data */
   2 space_left fixed bin (35),   /* sectors remaining */
   2 pad (15) fixed bin (35);     /* pad to sector boundary */
```

bk_disk_$init will perform consistency checking in bk_vol_header to assure that the header read from the pack is valid, that the pack is conditioned for data, and that no valid data already on the pack will be overwritten. The data patterns and form of checking must include the volume_id, unique_id, and first_clock values which will be matched with data safe-stored in the hierarchy. An appropriate place for the safe-storage is >ddd>backup>[volume_id]. For the dumps, >ddd>backup>[volume_id] will be created if it does not exist or is unreadable. For reload, the user will be queried if he desires to override the error and create >ddd>backup>[volume_id] or to remount with a different pack.

For the dumps, consistency checking will include a scan of the pack, starting with first_rec, using the "This is the beginning of a backup logical record" flag string and the word_count fields of the backup logical record header to follow the thread of entries on the pack. (See section 3. following for pack data format.)

This scan will avoid overwriting data in the case of a system crash with backup awake and unable to write bk_vol_header back out to the pack.

For each entry found whose first preamble record sector number is less than next_rec, backup_preamble_header.dtd must lie between first_clock and last_clock. If backup_preamble_header.dtd is earlier than first_clock, next_rec is in error and will be reset to the sector number of the first record of the entry and the scan stopped. If backup_preamble_header.dtd is later than last_clock, last_clock is in error and will be reset to the value of backup_preamble_header.dtd and the scan continued.

If entries are found whose first record sector number is greater than next_rec, they are va<id only if backup_preamble_header.dtd is later than last_clock. next_rec and last_clock will both be updated for such entries and the scan continued.

If the bk_vol_header data is inconsistent or indicates the pack is full, the user will be queried for reinitialization of the pack or mounting of another pack. In this context, "full" is defined as having less than 10,000 sectors remaining as given in space_left. The rationale for this definition is that a pack with only 625 pages remaining would probably have to be changed during the first backup pass.

Pack initialization consists of overwriting the pack header (and >ddd>backup>[volume_id]) with the volume_id as input by the user, an unique_id newly obtained from unique_chars_, first_clock and last_clock both to current clock_, first_rec and next_rec both to [header sector]+1, and space_left to [pack_capacity] - [header_space]. [pack_capacity] is currently 311600 for d191. Older secondary storage devices are not supported by ioi_ and rdisk_. [header_space] is currently 1, but may be more if a standard pack label becomes required.

For reload, bk_disk_$init will not scan the pack but will initialze an internal static variable, next_reload_rec, with the value of first_rec.

3. bk_output will be modified to check the -disk option flag in bk_ss_ and, if the option is taken, to call bk_disk_$write after the segment is successfully written to the tape.

bk_disk_$write will test the space required against the space remaining in space_left. If the space is insufficient, bk_vol_header will be updated with a new value of last_clock and written out to the header sector of the pack. After the header sector is overwritten, bk_disk_$write will set the "-init" option flag and call bk_disk$init for the mounting of a fresh pack. All packs after the first will be initialized.

when sufficient space is available, bk_disk_$write will write the preamble and the buffered segment data to the pack using iox_$rewrite_record and key values derived from next_rec. After the segment is successfully written, bk_disk$write will update next_rec and space_left and will write a trailer record containing the string "NEXT BACKUP RECORD" and the value of next_rec into sector <next_rec>.

The format of the pack is as follows...

```
 ______________________________
|                              |
| volume header                |  Initially sector 0.
|______________________________|
|                              |
| backup header & preamble 1   |  n sectors as required.
|______________________________|
|                              |
| segment 1                    |  p sectors as required.
|______________________________|
|                              |
  etc. to end of pack
```

4. backup_map_ will be modified to extract the volume_id from bk_vol_header and place it in the map with (or in place of, for a reload) the tape reel label information.

5. backup_load will be modified to query the -disk option flag in bk_ss_ and to call bk_disk_$read instead of bk_input if the option is taken. Note that this call to an entry point in >tools precludes the use of the disk for reloading of system libraries and leads to the limitation to reloads from backup and catchup prepared packs only.

bk_disk_$read will use iox_$read_record to read the backup header and preamble whose first record is given by the value of next_reload_rec as set by bk_disk_$init and will then apply the consistency criteria given for the pack scan in section 3 for the acceptance of the data. If the data is acceptable it will be read in and next_reload_rec <u>and</u> bk_vol_header will be updated as necessary. Reading will then proceed with the new value of next_reload_rec.

If the data is not acceptable and next_reload_rec is less than bk_vol_header.next_rec, the user will be advised of a data format error and queried concerning retry, search for next header, or quit. If the data is not acceptable and next_reload_rec is equal to or greater than bk_vol_header.next_rec, the user will be advised of the end of readable data and queried concerning additional packs to be reloaded. bk_vol_header will be rewritten to the header sector of the pack before the changepack order is issued in case it was updated during the reload.

6. end_dump will be modified to query the -disk option flag in bk_ss_ and, if the option is taken, to "close out" the pack and to demount it with a call to bk_disk_$release.

7. backup_cleanup will be modified to query the -disk option flag and to call bk_disk_$release if the flag is set.

D. Summary Description of bk_disk_.

The following are the declarations, usages, and functions of the bk_disk_ entries.

Entry

```
dcl bk_disk_$init entry;
```

Usage

```
call bk_disk_$init;
```

Called by start_dump, catchup_dump, and reload during initialization and by bk_disk_$write when current pack is full.

Functions

1. Check for current attachment; issue iox_$order changepack if true.
2. a. Attach drive with iox_$attach.
   b. If unattachable, query caller for retry or reset -disk option.
3. a. Read header sector into bk_vol_header (iox_$read_record, key "0").
   b. Verify bk_vol_header against >ddd>backup>[volume_id].
4. If reload command, set next_reload_rec to first_rec and return.

5. If pack is full (space_left ≤ 10000), call for fresh pack.
6. a. Chase entry thread on pack from first_rec to next_rec, verifying that backup_preamble_header.dtd lies between first_clock and last_clock.
   b. Chase entry thread beyond next_rec while backup_preamble_header.dtd is greater than last_clock, updating next_rec and last_clock, to the end of readable data.
7. Update >ddd>backup>[volume_id] and write bk_vol_header back to the header sector to retain the results of the scan.

Entry

    dcl bk_disk_$write entry (ptr, fixed bin, ptr, fixed bin);

Usage

    call bk_disk_$write (pream_ptr, pream_len, seg_ptr, seg_len);

Called by bk_output.

Functions

1. a. Check space required using word counts in pream_ptr -> preamble.
   b. If not enough space, update volume header and call bk_disk_$init for a fresh pack.
2. Write preamble and segment to pack using iox_$rewite_record and keys from next_rec.
3. Update space_left and next_rec for space used.
4. Write a single sector trailer record into next_rec containing the string "NEXT BACKUP RECORD", the value of next_rec, and the current value of clock_.

Entry

    dcl bk_disk_$read entry (ptr, fixed bin, ptr, fixed bin, fixed
        bin (35));

Usage

    call bk_disk_$read (pream_ptr, pream_len, seg_ptr, seg_len,
        error_code);

Called by backup_load.

Functions

1. Duplicates the functions of bk_input, substituting the disk pack for the tape reel and, in addition, applies the data acceptance criteria used by bk_disk_$init in scanning the pack.

Entry

```
dcl bk_disk_$release;
```

Usage

```
call bk_disk_$release;
```

Called by end_dump and backup_cleanup.

Functions

1. a. Queries disk option flag in bk_ss_, returning if the option is not taken.
   b. Makes a final update to bk_vol_header and >ddd>backup>[volume_id] and writes bk_vol_header back out to the header sector on the volume.
   c. Releases the channel with iox_$detach.

E. System Crash Recovery.

A system crash with backup asleep between passes has no effect on the consistency of the data on the disk pack since bk_disk_ is able to write a trailer record behind the last valid segment dumped. A trailer record is not possible on the tape because of the prohibition against backspacing and overwriting such a trailer with the next header.

A system crash with backup awake and writing causes the segment being dumped to be incomplete on either disk or tape. Normal practice calls for the writing of several EOFs on the backup tape before it is demounted so that reload will detect bad format (unexpected EOF) and will refuse to reload the incomplete segment. When reading from the disk, the data consistency check will refuse to reload the incomplete segment because it is not followed by either the next header or a disk trailer record.

F. Disk Error Handling Strategy.

Disk error handling must be considered differently during reading and writing.

In general, disk read errors will be handled similarly to tape reading errors. The read will be retried by rdisk_ according to its own strategy. If the read cannot be done, the reload of the segment being processed will be abandoned with a suitable entry made to the error file. If the full preamble or segment pages are being read, reload will skip immediately to the next header record and proceed with the next segment. If the first sector of a preamble header is being read, reload will proceed with 64-word reads searching for the next preamble header.

If an unrecoverable write error occurs, bk_disk_$write will abandon the space used so far for the segment being dumped, update next_rec for the space abandoned, and will overwrite the first 64-word sector of the preamble with the string "BAD SPACE HEADER" and the sector count of the bad space abandoned. It will then make a fresh attempt to write out the preamble and segment at the new value of next_rec. If the write error occurs during the writing of the first 64-word sector of the preamble, bk_disk_$write will overwrite the sector with the "BAD SPACE HEADER" and a sector count of zero and then attempt to write a header into the next 64-word sector. When reload detects the zero sector count or the unreadable 64-word sector, it will begin its search for a valid preamble header thus bypassing the bad space.