TO: Distribution

FROM: R. E. Mullen

DATE March 18, 1974

SUBJECT: Backup Performance

It has been known for some time that the Multics backup facility uses a large percentage of system resources, upwards of twenty percent.

This document is intended to provide an initial accounting of what resources are used and why. It is motivated by a desire to determine what inefficencies exist in backup and what steps can be taken to improve its performance significantly. In particular it is not intended at this time to redesign or rewrite backup, as proposed in MCB - 1076. In addition it is hoped that the operations interface and the format of the backup tapes will not need to be changed in order to improve performance. Indeed being able to show that backup works just as it did before (but faster) will provide a check against introduction of new bugs.

Overview

Considering backup's task of copying information to tape from directories and segments throughout the hierarchy it should be expected that backup places a heavy paging load on the system. Nevertheless as can be determined from monthly statistics on the Multics system at MIT backup has one of the lowest ratios of memory usage to cpu usage of all users. This suggests that the first step toward improving backup's performance is to determine how all that cpu time is being used.

On the other hand at times when response is bad, backup does not run (does not get scheduled because it is not an interacting user). Also if backup can be made to use 5 % less cpu time its paging load on the system will become a more significant factor.

Use of cpu time by backup

The following table gives an approximate breakdown of the percentage of its cpu time backup uses for various functions:

tape io	
copy segments	4
write tape	25
writing map	7
hcs_ calls	
for info	51
for effect	8
other	5

Although the figures above are percentages, they might usefully be thought of as minutes of cpu time on a busy day.

Tape io time

By running backup with the "-nooutput" option backup can be made to do everything it usually does except write the backup tape. In this case it uses about 25% less cpu time, so the inference is that tape writing accounts for 25% of backup's cpu time. It is assumed for the present that this time is irreducible, although significant improvements in tape io would have a beneficial effect on backup and thus on total system throughput.

Presently backup copies each non-directory segment to be dumped into a temporary, which is actually written to tape. This prevents backup from taking a segfault error should the user delete the segment or otherwise deny backup access after tape io has begun. However it has the disadvantage of increasing backup's working set considerably. A better solution to this problem would be to set up a handler for the segfault error which would do whatever is necessary to keep the tape consistant, and then proceed to process the next segment.

Map writing time

By running backup with the "-nomap" option, backup can be made to run as usual with the exception that a map of what has been dumped to tape is not written. In this case backup runs about 7% faster. It should be noted that entrynames are sorted in alphabetical order only if a map is being output, so the time to write the map includes the time to sort the entrynames in each

directory being dumped. As yet the time for the sort alone has not been measured. It is expected that the new formline and new vfile dim will reduce the map writing time considerably. Furthermore if the sorting operation turns out to be expensive it could be recoded to sort only those entry names corresponding to inferior segments or directories which must be dumped.

Time_in_hcs_

Looking at the code in backup it is apparent that aside from tape io and map writing the only other thing it does is call hcs_ a large number of times. A good measure of the time spent in these calls to ring zero can be gained from the meter_gate command. The assumption is made that backup is the primary user of hcs_$set_backup_dump_time and hcs_$list_dir. The number of calls to the former is a measure of the number of segments and directories dumped, and the latter is equal to the number of directories dumped, and by inference tells the number of nondirectory segments dumped. A close examination of the number of calls to other hcs_ entries which are used by backup is consistent with these assumptions.

The following table gives the average amount of time spent in various hcs_ calls made repeatedly by backup. (These times are averaged over all calls made on a given day by all users, however for all of those which are susceptible to large variations in execution time such as list_dir, backup is the primary user.) The cpu time spent on information gathering calls amounts to approximately 51% of backup's cpu time. Calls to hcs_ made for effect (marked with an asterisk) account for another 8% of backup's cpu time.

Gate entry	ms/call	calls/dir	calls/seg
list_dir	12.	1	0
list_inacl_seg	14	8	0
list_inacl_dir	14	8	0
quota_get	14	1	0
list_dir_acl	30	1	0
*terminate_file	17	1	0
get_author	16	1	1
status_seg_activity	14	1	1
get_bc_author	16	1	1
get_safety_sw	14	1	1
get_max_length	14	1	1
*set_backup_dump_time	15	1	1
list_acl	25	0	1
*initiate	19	0	1
*terminate_noname	6	0	1

Totaling the appropriate times it can be determined that for each directory dumped backup must spend 465 ms gathering information it needs from ring zero. The corresponding figure for nondirectory segments is 99 ms. While dumping incrementally as backup does during the day, about 40% of the branches dumped are directories and about 60% are nondirectory segments.

OPTIONS

Because the majority of backup's time is spent in ring zero, it is worthwhile to consider here three different ways in which such time can be reduced, and estimate the possible savings.

Two new hardcore interfaces

To remove the worst inefficencies either or both of the following entries might be added to hcs_ or some other gate:

hcs_$list_inacl_all

This entry would return both directory and segment initial acls for all rings. It would require about 14 ms, rather than the 14 x 2 x 8 currently required. It would save 21 ms for each directory dumped.

hcs_$status_for_backup

This entry would return all information currently returned by the hcs_ entries get_author, status_seg_activity, get_bc_author, get_safety_sw, and get_max_length. In order to provide for upwards

compatibility it should return a structure with a version number so that backup can determine easily when it has changed. Thus a useful primitive, which returns a non-trivial amount of information could then be expanded as needed without requiring the simultaneous installation of hardcore and outer ring software. This entry would require about 17 ms rather than the 74 ms currently required to gather the same information. It would save about 57 ms for each directory or segment dumped.

If these two entries or similar entries were added to hcs_, backup would spend about 2.. ms (rather than 465) in calls to hcs_ for each directory dumped, and about 42 ms (rather than 99) for each nondirectory segment dumoed. Together they would save about 27% of backup's total cpu time. If backup uses 20% of the system then the addition of these two entries would reduce the load on the system by 5%.

Similar or slightly greater improvement in performance could be obtained if entirely new star_list_ and status_ type primitives were designed which took into account the specific needs of backup in terms of specific data needed and in terms of high performance. Because such star_list_ and status_ primitives would provide a convenience for backup but not provide a significant performance improvement over the two primitives proposed above they are not pursued further here.

Copying_directory_out

Backup could be provided with a special interface which would move an intact and consistant copy of a directory to ring one. Information in the copy would then be stored on tape in a format similar or identical to the current format. The reloader would then remain independent of the structure of a directory. This would eliminate almost all other calls to ring zero for a saving of about 32% of backup's cpu time. Most of the remaining time spent in ring zero would be spent in ring one instead. Besides the apparent advantage of reducing total cpu usage this strategy offers the advantage of a more than proportional reduction in the time backup is running in ring zero, locking directories, and using the system_free_seg. In addition, such tools as were developed to deal with a copy of a directory might be useful to the dump analyzing programs.

However in order to use all this information backup would need to replicate some of directory control in ring one and would therefore become dependent on the structure of directories. Because such a primitive

could not be made available to ordinary users to generate carry tapes or tape archive either a new facility would have to be designed to fill their needs or a considerabe amount of vestigal code would need to be maintained in backup for the ordinary user's benefit.

Dump_raw_directory_to_tape

Each directory could be copied to ring one and then dumped with a minimum of processing to tape. Such a prooosal has been made in MCB - 1076, and is considered here as an example of the limiting case of almost no calls into ring zero.

Summary

It is apparent that there is a fixed overhead of about 14 ms associated with each call into ring zero to gain information from directory control given a pathname (because of ensuing calls to find_, fs_get$dir_mode, etc.) For the purooses of the following analysis it is assumed that any excess time is in some sense useful (eg. chasing through linked lists). As we have seen, adding two entries to hcs_ reduces overhead by reducing the number of calls into ring zero.

If a directory is copied out intact to an outer ring for information gathering then overhead will be further reduced but most of the useful work will still need to be performed if backup is to work more or less as it does currently.

If directories are dumped as raw bits onto tape then most of the "useful" work can be avoided.

The following table gives the times in ms per item dumped which might be expected for the various options considered above.

This table accounts for what would become of time currently spent in calls to hcs_ for information. It does not take into account any other factors such as changes in the amount of tape io if directories were dumoed raw to tape.

	current	2 hcs_	copyout	rawdump
Directories				
overhead	350	85	30	30
useful	115	115	115	15
total	465	200	145	45
Nondir segs				
overhead	84	28	5	5
useful	15	15	15	15
total	99	43	20	20
Savings				
backup's cpu time		27%	32%	41%
system cpu time		5%	6%	8%

APPENDICES

The following pages contain descriptions of the two new hcs_ entries proposed above.

Name: hcs_$list_inacl_all

This subroutine is used to list for all rings the initial Access Control List (Inital ACL) for new directories and the initial ACL for new segments. See the SWG write-ups for hcs_$add_dir_inacl_entries and hcs_$add_inacl_entries for descriptions of the seg_acl and dir_acl structures refered to below.

Usage

```
declare hcs_$list_inacl_all entry (char (*), ptr
     ptr, ptr, fixed bin (35));

call hcs_$list_inacl_all (dirname, area_ptr,
      area_ret_ptr, info_ptr, code);
```

1) dirname
   is the name of the directory in question. (Input)

2) area_ptr
   points to a user supplied area into which segment and directory initial acls will be placed. (Input)

3) area_ret_ptr
   points to allocated storage containing list of segment and directory initial acls. (Output)

4) info_ptr
   points to a structure, described below, which will be filled in with the locations and number of entries for each of the each of the lists of initial acls. (Input)

5) code
   is a standard status code. (Output)

Note

The argument info_ptr points to the following structure:

```
declare 1 inacl_info based (info_ptr) aligned,
          2 sia_relp (0:7) bit (18),
          2 sia_count (0:7) fixed bin,
          2 dia_relp (0:7) bit (18),
          2 dia_count (0:7) fixed bin;
```

1) sia_relp (i)

is a relative pointer (relative to ret_area_ptr) to the seg_acl structure listing the segment Initial ACL entries for ring (i).

2) sia_count (i)

is the number of entries in the seg_acl structure for ring (i).

3) dia_relp (i)

is a relative pointer to the dir_acl structure listing the directory initial ACL entries for ring (i).

4) dia_count (i)

is the number of entries in the dir_acl structure for ring (i).

Name: hcs_$status_for_backup

This suoroutine returns items of information about a segment which are not returned by other status or listing primitives.

Usage

```
declare hcs_$status_for_backup entry (char (*),
     char (*), ptr, fixed bin (35));

call hcs_$status_for_backup (dirname, ename,
     rptr, code);
```

1) dirname — is the pathname of the directory containing the entry of interest. (Input)

2) ename — is the entryname of interest. (Input)

3) rptr — points to a structure, described below, in which returned information is to be placed. (Input)

4) code — is a standard status code. (Output)

Note

The argument rptr points to the following structure:

```
declare 1 bstatus aligned based (rptr),
          2 version fixed bin,
          2 actime bit (36),
          2 actind fixed bin (35),
          2 max_length fixed bin (19),
          2 switches ,
            3 safety bit (1) unal,
            3 gate bit (1) unal,
            3 pad1 bit (34) unal,
          2 call_limiter bit (14) unal,
          2 pad2 bit (22) unal,
          2 pad3 bit (72),
          2 author char (32),
          2 bc_author char (32),
          2 entry_pad (3) bit (36),
          2 pad4 (5) bit (36);
```

1) version — is the version of this structure. If version = 1 then the above structure declaration is correct.

2) actime — is the file activity time.

3) actind — is the number of page faults taken on this segment.

4) max_length — is the maximum length to which the segment may grow in words.

5) switches

safety	if on, this segment may not be deleted.
gate	if on, the segment is a gate.

6) call_limiter — if the segment is a gate, all calls from other segments must be to a computed address less than call_limiter.

7) author — is the name of the user who created the segment.

8) bc_author — is the name of the user who last modified the bit count of the segment.

9) entry_pad — is the three words of padding at the end of a directory entry.