From: W. Olin Sibert

To: MTB Distribution

Date: October 24, 1980

Subject: Subroutines and Tools for Manipulating Disk Partitions.

This MTB describes subroutines and commands for accessing data stored in Disk Partitions. These are useful today as debugging utilities, and are also required for a contemplated redesign of Multics Initialization. Comments and questions should be directed to the author:

By Multics mail at MIT or System-M to:

Sibert.Multics

Or by mail to:

W. Olin Sibert
Cambridge Information Systems Laboratory
HED MA22

HVN 8*261-9353

Introduction:

A Multics File System Disk must contain at least three components: the Volume Header (label, allocation bit maps, dumper bit maps, etc.), the Volume Table Of Contents (VTOC), and the Paging Region. These are all required for use with the Storage System. In addition to these components, a disk (volume) may also have defined on it up to 47 "partitions". A partition is simply a contiguous region of disk, containing an integral number of records, which is not part of the volume header, VTOC, or paging region.

Presently, the supervisor makes use of several partitions for functions for which the normal file system cannot be used. The reason the normal file system cannot be used for these functions is that they must be available under circumstances where the integrity of the file system is not assured, and therefore these supervisor functions must manage their own disk space Examples of these functions are the LOG partition, used by the syserr logging daemon to buffer messages, the DUMP partition, used by BOS to write an FDUMP into, and the HC partition, used by the supervisor to dynamically page from. In addition, BOS uses the BOS partition to store the BOS commands and runcoms.

Today, there is no convenient way to manipulate partitions. The BOS DUMP, SAVE, and TEST commands can be used, and there are special-purpose functions in Multics (print_syserr_log, copy_fdump) which can access specific partitions. There is no general mechanism for partition I/O. Additionally, the only way to learn what partitions are on a particular volume is to dump the label in BOS, and interpret it by hand, or salvage the volume and look at the summary report.

This MTB describes a set of subroutine interfaces and commands which use them to perform various useful functions. Commands are available to list, dump, and clear partitions; the subroutine interfaces provide an interface to the partition such that it appears as a continuous array of words. A subroutine interface for examining a volume label is also provided.

The Implementation:

This implementation described is designed to be adequate without being overcomplicated. Various niceties such as efficiency and administrative control have been largely omitted, because the need for them does not seem very great. These interface will not be used except by system administrators or system programmers, and not very often at that; therefore, it does not seem worthwhile to implement complex control or performance mechanisms.

The read/write interfaces are in the hphcs_ gate, because there is no access control function enforced by the primitives. There is no locking or protection against simultaneous updates. Any control to be exercised over the access to partitions must be done in an outer ring, or manually by the users.

In view of the expected uses of this mechanism, the lack of locking and simultaneous update protection does not seem very important. The mechanism was implemented primarily to allow Multics System Tapes to be written into disk partitions (for purposes of subsequent booting therefrom), and this is an operation which will be performed rarely at most sites, and always by some one person or closely coordinated group of people who can be trusted not to destructively interfere. The only other anticipated use of the writing interface is for performing system repair or maintenance operations, and this, too, is something which will be the province of a very small number of people.

The volume label reading interface is in phcs_ because it seems consistent with the concept of phcs_ that a user with access to it should be able to look at all "system data", of which volume labels are certainly a form. The partition read interface is not in phcs_ because it provides unrestricted access to data in any disk partition, which may (though no partition does so now) contain "user data", which should not be available through phcs_.

The read/write mechsnism is not very efficient. It requires two disk I/Os for each page accessed, and two more for each call to the primitives; therefore, to read a page at a time with the primitives requires four I/Os per page. Worse yet, the two I/Os per call are to the volume label, which may be on the opposite end of the disk from the partition, since many partitions are allocated from the far end. This is not a problem; none of the uses for these subroutines requires high speed.

Possible Future Enhancements:

The major missing features of this implementation are selective access control, (primitive access control is, of course, provided by the acl on hphcs_) locking, and efficiency. None of them would be very complicated to implement; if the interface turns out to have wider applications than initially envisioned and the enhancements become more desirable, they should certainly be implemented.

Access control could be implemented by access control segments. The ACS's for a volume would be located in a directory whose name is the PV name, and which itself would be located either in >lv or some subdirectory thereof. The ACS's and containing directories should be created automatically by ring one volume management at volume registration or rebuild time. Access could be controlled either by the primitives themselves in ring zero, or by another level of interface in ring one.

Locking could also be implemented either in ring zero or ring one; because there are no present applications for either, it is hard to say which would be most appropriate. A database somewhat like the dirlock table would be used to implement locking. Presently, the read/write interfaces are protected against volume demounting occurring in any single call to the primitives, but that is as sophisticated as it gets.

The primitives today work by calling read_disk and write_disk. These are page-at-a-time interfaces which are not very efficient to begin with. Additionally, the primitives must read the volume label once per call, in order to locate the desired partition. Efficiency could be improved dramatically if the primitives were to construct their own page tables, in the manner of copy_*fdump*, and read/write all the requested records at once. Another possible improvement would be to have some mechanism for specifying partitions which did not require reading the volume label each time to locate them and verify the user supplied addresses.

Applications:

The primary application (that is, the reason they were implemented) for these interfaces is to provide a way to write MST files into disk partitions to experiment with boot-from-disk technology. This has been incorporated into a command, write_mst_partition, which takes a file created by generate_mst -file and writes it into a specified disk partition. The command is not described in this MTB because it is not part of the general interface, and is still under development.

The list_partitions command and the read label subroutine answer a need expressed by several SiteSAs that there should be some convenient way to get at this information. It is also useful when developing any application which uses disk partitions.

The dump_partition and clear_partition commands are primarily useful for developing applications which use disk partitions. They can also be used to examine problems with existing applications, such as garbled FDUMPs and whatnot.

A tool could be written to allow the BOS directory to be updated from Multics, thus saving much of the bother of editing the CONFIG deck or runcom, generating a new tape, etc.

If the efficiency problems are remedied, it would be useful to rewrite copy_fdump to be a largely user-ring program, so that it could be modified more readily for new and different FDUMP formats.

Interface Documentation:

Documentation for the following commands and subroutines follows; they are all intended to be documented in Tools (AZ03), save for mdc_$pvname_info, which belongs in the SWG, and is documented in this MTB only because it is useful, and not presently documented anywhere else.

list_partitions
dump_partition
clear_partition

phcs_$read_disk_label
mdc_$pvname_info
hphcs_$read_partition
hphcs_$write_partition

10/19/80: list_partitions

Function: lists the locations and sizes of all the partitions on a specified physical volume.

Syntax: list_partitions Pvname

Arguments:
Pvname
The name of the physical volume whose partitions are to be listed.

Access required:
Access to the phcs_ gate is required.

Output format:
The output consists of a header, which lists the physical and logical volume names, the PVID and LVID, the size of the volume in pages, the size of the VTOC in both pages and VTOCEs, the size of the paging region, and the number of partitions. It is followed by a table listing the name, first record, and size of all partitions and other regions on the volume. All numbers in the table are given in both decimal and octal (in parentheses), and all other numbers in the output are decimal.

Example:
```
Volume root2 (740651611731) of logical volume root (225072707470):
 38258. total records.  2000. VTOC records, for 10000. VTOCEs.

Volume map (including 4 partitions):
 Name               First record          Size
Volume header           0. (0)              5. (5)
VTOC area               5. (5)           2000. (3720)
  BOS                2005. (3725)         200. (310)
  DUMP               2205. (4235)        2000. (3720)
Paging region        4205. (10155)      34712. (103630)
  HC                36917. (110065)      1200. (2260)
  ALT               38117. (112345)       141. (215)
```

10/19/80: dump_partition

Function: displays data from a named disk partition; by default in octal with four words per line, though other output formats can also be selected.

Syntax: dump_partition Pvname Partname Offset {Length} {-control_args}

Arguments:
Pvname
   The name of the physical volume on which the partition to be dumped exists.
Partname
   The name of the partition to be dumped. It must be four characters or less in length.
Offset
   The offset at which to begin dumping.
Length
   The number of words to be dumped. If not supplied, one word is dumped.

Control arguments:
-short, -sh
   Outputs data in short form, similar to dump_segment -short.
-long, -lg
   Outputs data in long form, similar to dump_segment -long.
-character, -ch
   Outputs data including the ASCII character representation.
-bcd
   Outputs data including the BCD character representation.
-no_header, -nhe
   Suppresses the header.
-header, -he
   Prints a header. (Default)

Access required:
Access to the hphcs_ gate is required.

Function as an active function: returns the contents of the specified words in octal, separated by spaces, rather than printing them.

Syntax as an active function:
   [dump_partition Pvname Partname Offset {Length}]

10/22/80: clear_partition

Function: overwrites the contents of a disk partition with zeros or an optional user-supplied pattern word.

Syntax: clear_partition Pvname Partname {-control_args}

Arguments:
Pvname
   The name of the physical volume on which the partition to be cleared exists.
Partname
   The name of the partition to be cleared. It must be four characters or less in length.

Control arguments:
-brief, -bf
   Produces brief format messages.
-long, -lg
   Produces longer messages. (Default)
-pattern WORD
   Overwrites the partition with data consisting of the specified octal pattern word. The specified word is written into every location in the partition. If this control argument is not specified, a default of all zeros is used.

Access required:
Access to the hphcs_ gate is required.

Notes:
The user is always queried as to whether the partition should be overwritten; by default (if -brief was not specified), the contents of the first eight words in the partition are displayed (in octal and as ASCII characters) as part of this question, to aid in preventing accidental overwriting of the wrong partition.

Entry: phcs_$read_disk_label

This entrypoint is used to read the label of a storage system disk drive. The label is described by the structure "label", in the include file fs_vol_label.incl.pl1.

Usage:

```
     dcl  phcs_$read_disk_label entry
          (bit (36) aligned, pointer, fixed bin (35));

     call phcs_$read_disk_label (pvid, label_ptr, code);
```

Arguments:

1) pvid (Input)
   is the physical volume id of the disk whose label is to be read. The physical volume id is used instead of the volume name because this is a ring zero interface, and volume names are not accessable by ring zero; hence, all ring zero interfaces which refernce physical volumes use the pvid. A pvname may be converted to a pvid by calling the subroutine mdc_$find_volname.

2) label_ptr (Input)
   is a pointer to the user-supplied area in which to read the label. The label is 1024 words long, and is described in fs_vol_label.incl.pl1

3) code (Output)
   is a non-standard status code. It will be one of the following:

   zero
      indicates that the label was successfully read.

   error_table_$pvid_not_found
      indicates that the specified physical volume is not presently mounted.

   an integer between 1 and 10
      indicates that a physical disk error occurred while trying to read the label. Error messages for physical disk errors are declared in the include file fsdisk_errors.incl.pl1, in the array fsdisk_error_message.

Entry: mdc_$pvname_info

This entrypoint is get various information about a specified storage system physical volume.

Usage:

```
     dcl  mdc_$pvname_info entry (char (*), bit (36) aligned,
          char (*), bit (36) aligned, fixed bin, fixed bin (35));

     call mdc_$pvname_info (pvname, pvid,
          lvname, lvid, device_type, code);
```

Arguments:

1) pvname (Input)
is the name of the physical volume about which information is to be returned.

2) pvid (Output)
is the physical volume id of the specified volume. It can be used as a parameter to ring zero volume and partition interfaces.

3) lvname (Output)
is the name of the logical volume to which the physical volume belongs.

4) lvid (Output)
is the logical volume id of the logical volume to which the physical volume belongs.

5) device_type (Output)
is a number indicating what type of device the specified physical volume is mounted on. The names and characteristics of these devices are listed in various arrays declared in the include file fs_dev_types.incl.pl1.

6) code (Output)
is a standard system status code. It is non-zero if the information about the volume cannot be obtained, or if the volume does not exist.

Entry: hphcs_$read_partition

This entrypoint is used to read words of data from a specified disk partition on some mounted physical storage system disk.

Usage:

```
     dcl  hphcs_$read_partition entry (bit (36) aligned, char (*),
          fixed bin (35), pointer, fixed bin (18), fixed bin (35));

     call hphcs_$read_partition (pvid, partition_name,
          offset, data_pointer, word_count, code);
```

Arguments:

1) pvid (Input)
is the physical volume id of the disk from which to read. The physical volume id is used instead of the volume name because this is a ring zero interface, and volume names are not accessable by ring zero; hence, all ring zero interfaces which refernce physical volumes use the pvid. A pvname may be converted to a pvid by calling the subroutine mdc_$find_volname.

2) partition_name (Input)
is the name of the disk partition to be read from. It must be four characters long or shorter.

3) offset (Input)
is the offset in words, from the first word of the partition, of the first location to be read. It must be non-negative, and less than the number of words in the partition.

4) data_ptr (Input)
is a pointer to the user supplied area into which the data is to be read. It must be aligned on a word boundary.

5) word_count (Input)
is the number of words to be read. The sum of offset and word_count must be less than or egual to the number of words in the partition. The word_count must also be less than or equal to sys_info$max_seg_size.

6) code (Output)
is a non-standard status code. It will be one of the following:

zero
indicates that the data was successfully read.

error_table_$pvid_not_found
indicates that the specified physical volume is not presently mounted.

error_table_$entry_not_found
indicates that the specified partition could not be found.

error_table_$out_of_bounds
indicates that read request attempts to access data outside the partition; that is, the sum of offset and word_count is too large.

an integer between 1 and 10
indicates that a physical disk error occurred while trying to read the label. Error messages for physical disk errors are declared in the include file fsdisk_errors.incl.pl1, in the array fsdisk_error_message.

Entry: hphcs_$write_partition

This entrypoint is used to write words of data into a specified disk partition on some mounted physical storage system disk. No protection is provided against simultaneous use of ths entrypoint by several processes writing to the same partition; thus, care must be exercised when using it.

Usage:

```
     dcl  hphcs_$write_partition entry (bit (36) aligned, char (*),
          fixed bin (35), pointer, fixed bin (18), fixed bin (35));

     call hphcs_$write_partition (pvid, partition_name,
          offset, data_pointer, word_count, code);
```

Arguments:

All arguments are the same as for hphcs_$read_partition, including the possible values for the status code. The only difference is that data is written into the partition from the user-supplied area, rather than being read.