bcc	title	prefix/class-number.revision
	MEMORY MANAGEMENT	MM1/W-1

checked	authors	approval date	revision date
Butler	R. R. Van Tuyl	5/9/69	
checked		classification	
		Working Paper	
approved		distribution	pages
Mel		Company Private	25

ABSTRACT and CONTENTS

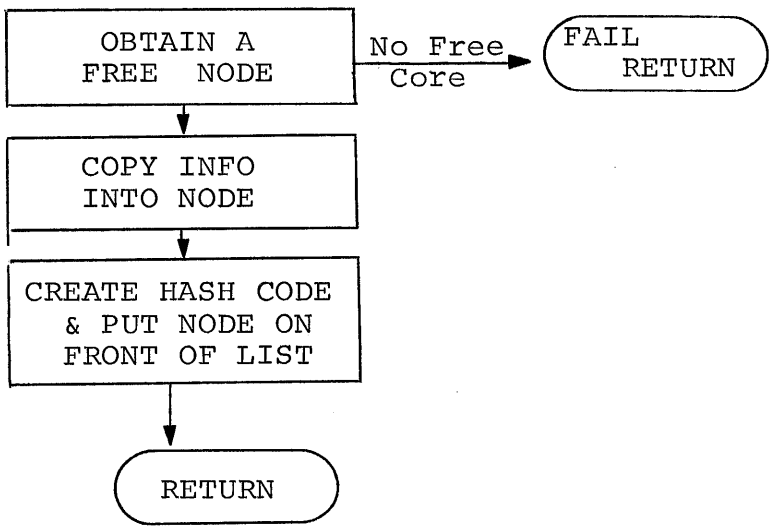
A description of the basic table, mechanisms, and interfaces of the swapper. This document will be very difficult to change after 30 April.

A page may appear on more than one device. The current version of the page may be on one or all of the devices. The Memory Management (MM) tables allow one to find the current incarnation of a page.

The Core Hash Table (CHT) is a bucket style, chained Hash Table. The table is composed of two parts, CHT1 and CHT2. CHT1 will reside in physical core between 100 and 477B. CHT1 is composed of 256 entries of one word each for efficient hashing. The entry is a pointer into CHT2.

CHT2 is composed of six word nodes formatted as depicted in Table 1. CHT2's size will be a parameter and depend on the amount of physical core in the main memory.

A free core list will be maintained through the CHT2 entries. The index of each CHT2 entry is the page number of the entry. The entry will also contain this page number. When an entry is made, the algorithm is as follows:



The CHT collision lists are of type 3 (see appendix). This is to facilitate removal of nodes from the free core list.

CHT1 is entered by hashing the class code. The following program will calculate a pointer into CHT1:

```
FUNCTION CHT HASH (UNØ, UN1);
     T ←— UNØ EOR UN1
     RETURN ( (T LCY 8) EOR (T RCY 8) EOR T EOR 264B)
          AND 377B + CHTBA;
```

Microcode Hashing Algorithm for CHT

UN in M&Q, result in RØ, and fetch begun.

```
Addr   MXQ,,(Q,R1); Move =132000B,M;
Cycle  MXQ,M,8; Move =CHTBA,Z
Cycle  MXQ,M,8; Move R1,Q; Move =77777400B;
       Bit TCONX;  GOTO,SØ,DR
Addr   MXQ,Z,RØ; PFETCH;
```

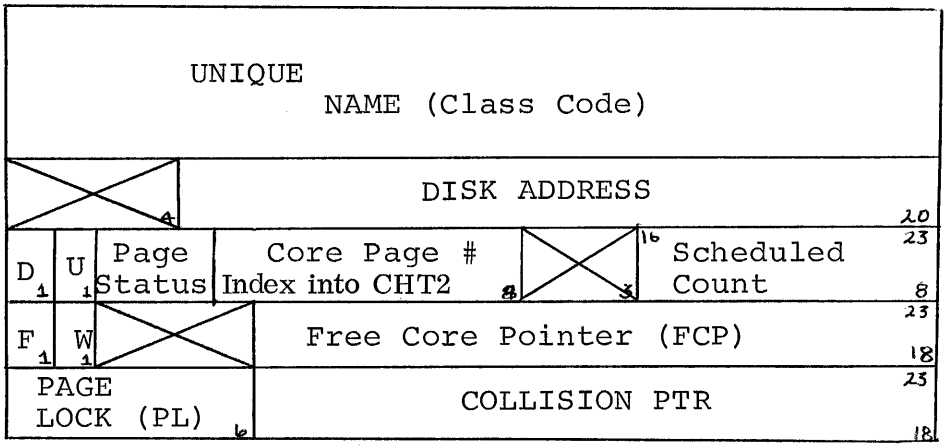
TABLE 1

CHT1 Entry

18-bit ptr into CHT2

CHT2 Entry

Hash (Unique Name) gives CHT1 entry



Core page address is given by index of entry

D - page must be put back onto drum.

U - Unavailable page

Page Status

ØØØ - Illegal

ØØ1 - Read from drum in progress

Ø1Ø - Write from core onto drum in progress

Ø11 - Illegal

1ØØ - Illegal

1Ø1 - Read Error

11Ø - Write Error

111 - Illegal

Scheduled Count - incremented once for each entry in CWS of a process which points to the page in core.

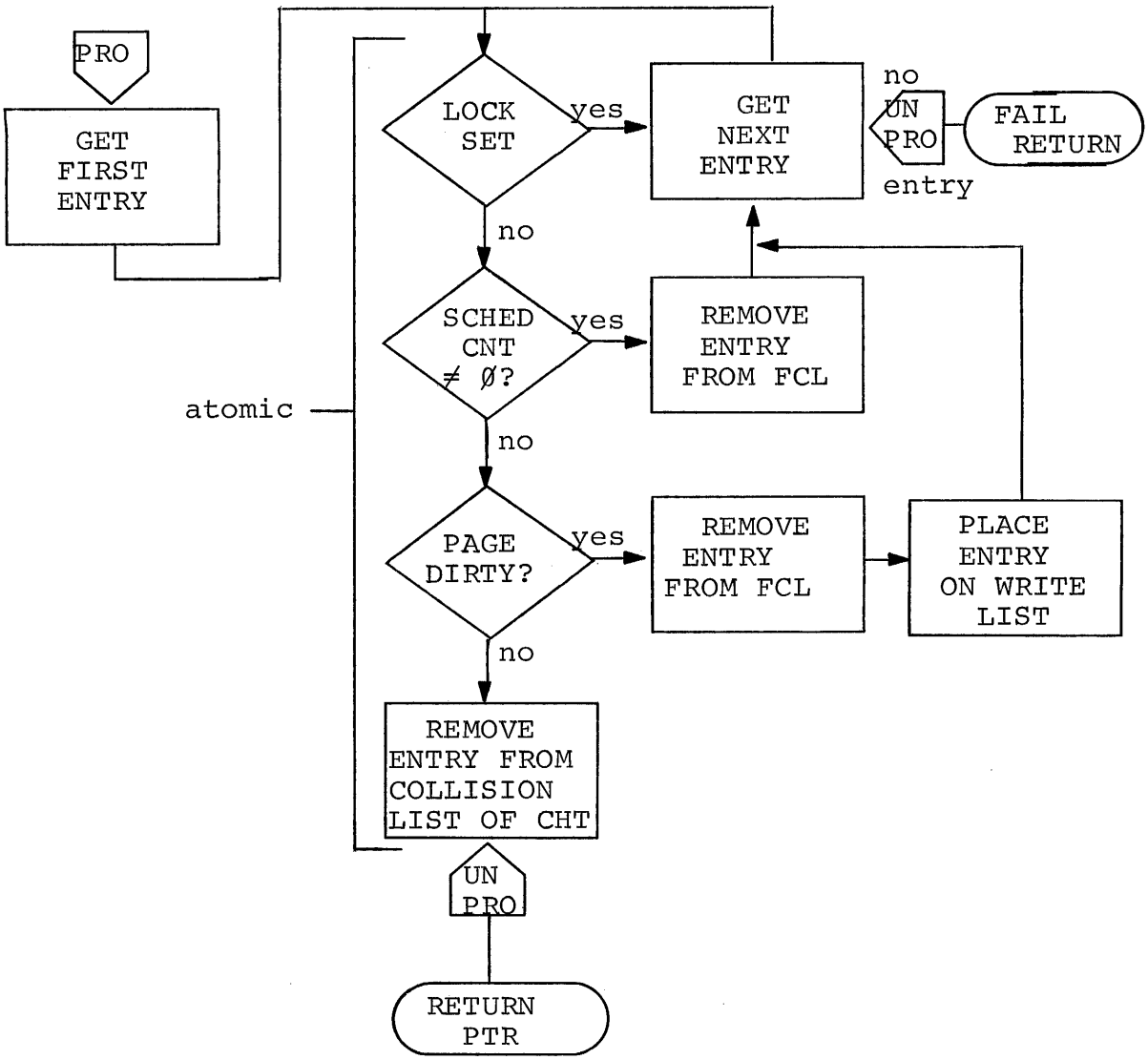
Page Lock - Bits set by privileged processes to lock the page into core.

F - FCP is a free list ptr

W - Page is on the write list

Removal of Entry on Free Core List

In the event a page is required, it is taken off the FCL. However, between the time it was placed on the FCL and the time it is to be removed it may be placed in use, or it may be necessary to write the page out, or it may be locked into core. In any of these cases, the swapper must take the appropriate action.



Drum Hash Table Specifications

The DHT is a large table which is entered by hashing a disk address. Its entries then determine the other place the page is located (either drum or fast disk area [FDA]).

The table has one entry for each page on the drum or fast disk. The design is for a 4 to 6 million word drum(s), which is 2000 to 3000 sectors. The table size will be parameterized on the size of drum.

The hashing scheme is as follows:

This algorithm assumes a disk address of 20 bits, and is mod any size table. The DHT table should be about 15% larger than necessary to limit the number of probes required.

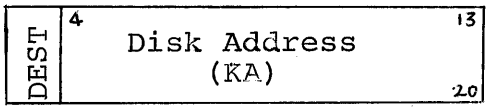
```
Function DHTHASH(KA);
T1 ◄- (T ◄- KA EOR (KA RCY 8) EOR 3152B) AND
   7777B) -DHTSZ;
RETURN T IF T1 ≥ Ø;
RETURN T1;
```

DHT will be split into two parts, DHT1 and DHT2. DHT1 will contain the disk address. This will allow efficient scanning for a free slot. DHT2 will contain two words which are indexed by the same index as the DHT1 word. This allows a simple LSH to do the multiply.

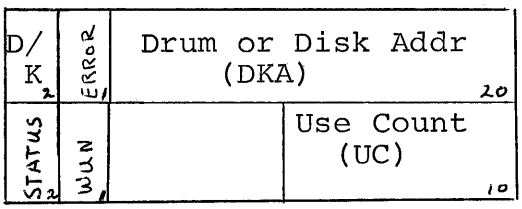
The collision strategy is to linearly scan forward to a free word (Ø) in DHT1. When a free word is found, the corresponding DHT2 entry is used. When an entry is deleted, all entries between the deleted entry and a free space must be rehashed and moved. This should be done infrequently and only for about 2 moved entries/deletions.

DHT Entry (split into two tables)

DHT1



DHT2



Where:

Hash (Disk address) gives DHT entry

KA — disk address of page

DKA — current drum address or disk address (fast disk)

DEST — page is being destroyed

STATUS
- ØØ new page, not on either address
- Ø1 on DKA only
- 1Ø on KA only
- 11 on KA & DKA both

UC	use count. When this goes to zero page may be deleted from drum or fast disk
D/K	ØØ no address
	1Ø fast disk address
	11 drum address
	Ø1 no address
WUN	Write Unique Name onto disk
ERROR	Read error

REQUESTS WHICH CAN BE MADE TO SWAPPER

The mechanism for making a request to the AMC is to copy the description of the request into a free node obtained from a shared Free List and leaving it on the approrpiate queue. The queue pointers will reside in physical core between 40 64.

Types of requests which can be made:

a. Bring process into core:
   1. Ptr to next entry on queue
   2. Ptr to PRT entry for process

b. Write process onto drum:
   1. Ptr to next entry on queue
   2. Ptr to PRT entry for process
   3. Mandatory or convenient write (latter for maintaining at least one process in core if swapper gets bogged down)
   4. Pages put on top or bottom of free list (to extend the "life" of special pages in core).

c. Return page to drum
   1. Ptr to next entry on queue
   2. Unique Name of page.

d. Transfer page from "drum" to disk and delete DHT entry if Use Count = Ø;

1. Ptr to next entry on queue
2. Ptr to PRT for process making request
3. Real Name of page
4. Check class code or check class code for Ø and write class code
5. Wakeup condition
    a. no wakeup
    b. wakeup when transfer completed

e. Transfer page from disk to "drum."

1. Ptr to next entry on queue
2. Ptr to PRT
3. Real Name of page
4. Wakeup condition
    a. seek begun (pages put on bottom of free list)
    b. transfer completed
    c. no wakeup
5. Pages put on top or bottom of free list

f. Write Unique Name (UN $\neq$ Ø) on free page on disk.

1. Ptr to next entry on queue
2. Ptr to PRT
3. Real Name of page

g. Write Unique Name on disk page with given class code (this is a special case not to be used generally in the system).

1. Ptr to next entry on queue
2. Ptr to PRT
3. Real Name to be written
4. Class code expected on the disk

h. Delete information and class code at given Real Name. Entire page set to Ø.

1. Ptr to next entry on queue
2. Ptr to PRT
3. Real Name of page to be deleted.

i. Disk to disk transfer

1. Ptr to next request
2. Old Real Name
3. New Real Name
4. Ptr to PRT
5. Wakeup condition

j. Request new page

1. Ptr to next request
2. Real Name of new page
3. PRT pointer

Wakeup generation by AMC

a. does not apply - process is placed on in core queue when working set is completely in core.

b. If the write is at the convenience of the swapper a wakeup is generated, when the write is started. When a swapper receives a process which is to be written at its convenience the unscheduler does not know if it will be run again. Therefore, if the swapper writes it out, it must be sure the scheduler will consider the process again. The swapper does this by generating a wakeup for the process.

c. Specified.

d. Specified.

e. Wakeup when operation completed.

f. Wakeup when operation completed.

g. Wakeup when operation completed.

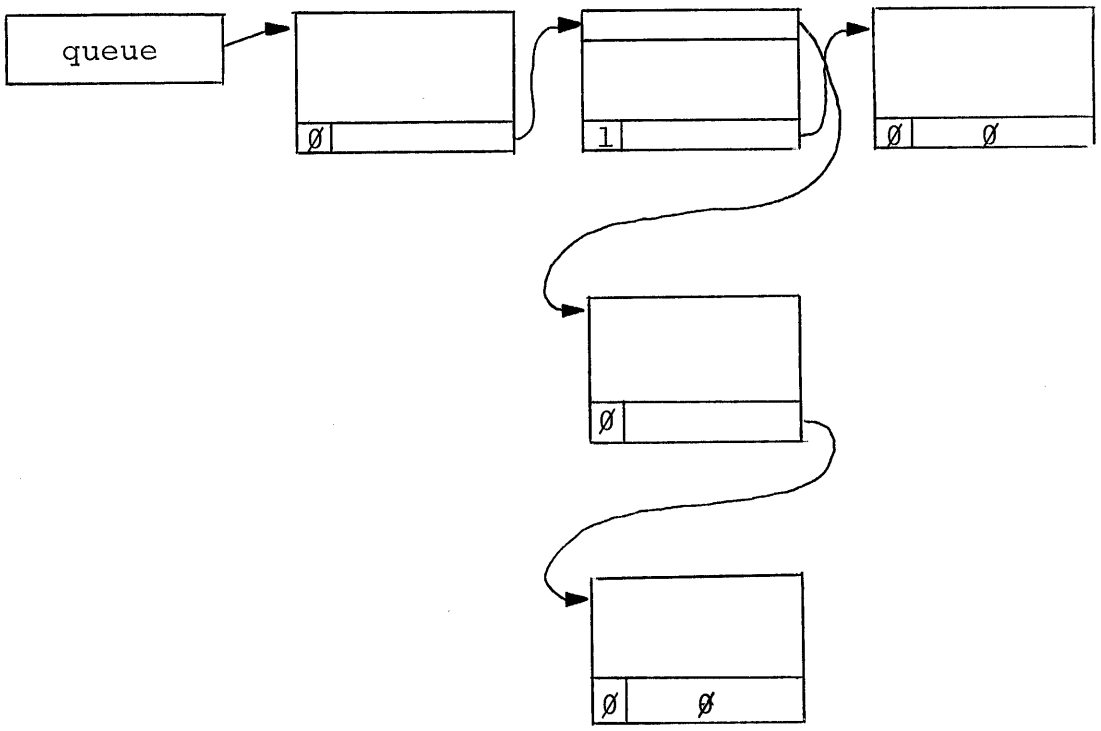
The following structure will allow several requests to be treated as a group. The first node to be spliced into the queue at the end will have two pointers:

a. Ptr to next entry in queue

b. Bit in ptr word to specify group

c. Ptr to first entry of group

All lists will end with a zero value for the pointer.



If a wakeup is desired at end of request, the AMC will set a count for the number of operations which will be performed and cause a wakeup when the count goes to zero.

Detailed Discussion of Each Request

A. Bring Process into Core.

1) Take request from request queue. If the Process in Core (PIC) bit is set, get the PRT entry and go to 3)a). Otherwise, place the request on the appropriate Drum Sector Read List (DSRL) to read the process's context block if the page is not in core. If the page is in core do No. 2.

2) When context block is read, the following occurs:

a) The PRT entry is then chained on the end of the Context Block Queue (CBQ).

b) If the convenient bit is on in the request, an unidentified algorithm will be used to determine if the process is brought into core. If the process is not, a wakeup is generated and the process is written out of core.

3) When the swapper requires more reads, it will remove a PRT entry from the front of the CBQ. The CWS will be scanned and the following occurs:

a) If the SF bit is set continue scan.

b) Otherwise, set the SF bit and do:

1. If the PIC bit is set do:

a. If the page is not in core (a page is considered not to be in core if the Status is Read in Progress) queue the read such that the Process is not put on QIC.

b. If the page is in core increment the Scheduled Count (SC) in the CHT for the page.

2. Otherwise

   a. If the page is not in core increment the Read Count in the PRT and queue the read such that when the Read Count goes to zero the process is put on QIC.

   b. Otherwise, increment SC for the page.

At the end of the scan the PIC bit will be set. If during the scan it is discovered that a page is on the disk:

a) If the PIC bit is set, the page is ignored.

b) Otherwise, the process will be removed from the DSRL and written back onto the drum. All the disk reads in the CWS will be queued with a wakeup to be generated when the seek for the last cylinder ends.

4) Each time a read is begun the PRT is checked to make sure that the PIC bit is set. If it is not, the read is discarded. If it is, a search of CHT will be performed.

a) If the page is found the SC count will be incremented. The count in the PRT will be decremented. If it goes to zero, the process PRT entry will be put on QIC.

b) Otherwise, the read request to the TSU begins.

   1. A free page is found on Free Core List (FCL) via the algorithm for getting free pages.

   2. The old entry is deleted and the new entry inserted. (This requires rehashing the old UN

and patching the collision chain.) The Read in Progress Status bits are set. The entry is then placed in CHT2. Its SC is set to Ø.

3. The read commands are sent to the TSU. The request node will be pointed to by a Read Cleanup Pointer (RCP).

5) When the read is completed, the following checks are made:

a) The TSU error word is checked. If an error occurred, there are two cases:

1. It is the first error, in which case the error and unavailable bits in the CHT entry are set and the page put on the free list. Exit.
2. It is the second error. Set error and unavailable bits on page and proceed to b).

b) The Unique Name will be checked against the class code in the request node (which was copied from the context block). If the UN's do not match, clear UN in CHT and put page on free list. Set BADUN bit in CWS entry for the page. Exit.

c) Check PRT to make sure the PIC bit is set. If not, put the page on the free list if SC = Ø.

d) We are now convinced that we want the page. Increment SC.

e) Scan the read sector list for additional entries with same Unique Name. Remove each such entry and do c) and d) for it.

B. Write Process onto Drum.

1) If convenient bit set, put process on Convenient Write Queue (CWQ) and exit.

2) For each page in the CWS (with non-zero PMT entry) of the process do:

a) If page not in CHT, consider next page.

b) Reset SF in PMT and decrement Scheduled Count (SC) in CHT. (If page destroyed, SF is set but page not in CHT.)

c) If (SC = Ø?) and Dirty Bit = 1 queue write for the page.

d) If SC = Ø and Dirty Bit = Ø, put CHT entry on FCL.

3) When it is determined to do a write, the following actions occur:

a) Get a write request off the Write List

b) If (SC V PL) = Ø, then reset the CHT dirty bit, set status in CHT to Write in Progress, place request node on cleanup pointer and send commands to TSU. Also zero the DHT drum address and replace old drum address in Drum Free Page Bit Table (DFPBT).

c) Otherwise, if PL ≠ Ø, put request on end of write list. Otherwise, return the request to free storage and go to a).

4) At the end of the write, the following actions occur:

a) The error status of the write is checked.

1. If an error occurred, replace the write node on the write list and set the dirty bit.

b) If PL $\neq \emptyset$, put node on end of write list. Otherwise, if (SC V D) = $\emptyset$, put page on FCL. Update DHT drum address and set status DKA bit and reset KA bit.

C. Return page to drum. This is to be used to cause SC to be decremented and the page written onto drum. There are two steps.

1) Decrement SC

2) If SC $\leq \emptyset$, do

a) If dirty bit set, queue write

b) Otherwise put page on FCL

D. Transfer page from drum to disk. The request is made with an activate.

1) Remove each entry on Request queue and check the use count in DHT. If the use count is >1, success return. Otherwise:

a) If the page is in core and if dirty bit in CHT entry set, go to b)1).

b) Otherwise, if the DHT status reflects that the page is not on the disk, but is on the drum:

1. Place the request on the appropriate Disk Cylinder Queue (KCQ) and fail return the activate.

c) If the page is not on the drum, generate a success return and exit.

Note: a success return is made if no transfer is required to satisfy the request.

2) When the seek begins for a cylinder, the KCQ is reformed onto the Disk Sector Source Queues (KSSQ). There are 6 KSSQ's. Also, the Disk Cleanup Pointer (KCP) is cleared.

Note: The following description applies to reads as well as writes.

3) When the seek ends, the first entry on each KSSQ is removed. If the entry has the same disk address as the entry on KCP, return the entry to KSSQ.

   a) If it is a drum to disk transfer, decrement the use count. If the use count is zero, the the following, other ignore the request.

      1. If the class code is to be written, put the request on the Disk Sector Action (KSAQ) for the sector involved.
      2. If the page is in core, increment SC and put the request on the proper KSAQ.
      3. Otherwise, setup the request as a drum-to-core transfer, and put into the microcode branch address field the address of a routine which will move the request onto the proper KSAQ after it has come in.

   b) If it is a disk to drum transfer, it is put on KSAQ.

4) At each time when another disk transfer is to be given to the TSU, an inspection is made of the KSAQ for the current disk sector.

b) Otherwise, the entry is removed. If UC ≠ Ø, we ignore the request. If UC = Ø the entry is placed on the Disk Cleanup Pointer (KCP). The appropriate commands are issued to the TSU. The KSSQ for the sector is inspected and if an entry exists, it is removed and treated according to 3a, b above.

5) When the disk write is completed, the following occurs for the entry on KCP and each entry on KSSQ and KSAQ with the same disk address:

a) If the request is to write the class code (this can result from F or from a DHT entry WUN bit being set) and no error was made, then do:

1. If the class code was zero, do

a. If the page is to be transferred, generate a drum to core transfer with a branch into the routine to put request node on KSAQ.

b. Otherwise, put node on KSAQ changing operation to write class code.

2. If the class code was not zero, do:

a. Leave note in swapper error word in PRT.

b. Generate wakeup (cause punt).

b) Otherwise, check for errors. If error was made, increment error count.

1. If Error Count > N, record disk address in Swapper Error Word (SEW) in PRT if SEW = Ø. If SEW ≠ Ø, set Swapper Error Waiting Word in PRT and store request node on the Swapper Error Queue (SEQ). A central process is

responsible for maintaining the SEQ.

c) Otherwise

1. Decrement SC and return CHT entry to FCL if SC = Ø.
2. Delete DHT entry if UC = Ø.
3. Decrement disk transfer field in PRT and generate wakeup if zero.

E. Transfer page from disk to drum. This request is made with an activate.

1) Remove entry from request queue. Find the current incarnation of the page.

a) If page already in core or on drum (find out by looking in DHT and checking status), ignore request and success return from the activate.

b) Otherwise, put the request on the KCQ and fail return activate.

2) When the request is removed from KSSQ (refer to D for actions common to reads and writes), the following actions occur:

a) Check the current incarnation of the page. If it is anywhere besides the disk, ignore the request.

b) Put the request on KSAQ.

3) The transfer is caused when the entry is removed from the KSAQ. The entry is pointed to by KCP.

4) When the transfer is completed

a) If there are errors, the error count field is incremented and the read requeued. Exit.

b) The request is put on the DWL.

F. Write Unique Name on Deleted (Empty) Page on Disk.

1) Remove request from queue.

2) Queue a class code read on KCQ.

3) When read takes place

a) If error, requeue class code read and exit.

b) If class code not zero, have a message in SEW for the process if SEW = Ø, and exit.

c) Otherwise, queue class code write.

4) When class code write is finished, check errors. Attempt to write n times and generate interrupt if writes unsuccessful.

G. Delete Information and Class Code at Given Real Name.

1) Remove entry from request queue.

2) Find page in DHT. Set destroyed bit. If no entry exists, make one. Note: In disk and drum reads and disk writes, if destroyed bit is set, the operation must be aborted.

3) Find entry in CHT.

a) If in CHT

1. Set unavailable bit.

2. Remove page from CHT if clean - zeroing class code and disk address.

4) Flush map of both CPU's

5) Queue class code check on Disk Cylider Queue if DHT CC bit claims class code written on disk.

a) If class code checks ok, queue class code write of Ø and transfer Ø to entire page.

b) If class code does not check, remove destroyed bit from DHT entry and exit.

6) Remove DHT entry and return drum page to free drum bit table.

H. Disk to fast disk transfer

Same as disk to drum transfer, except that the sector on which the page is written is picked from the current cylinder on the fast disk. If that side is seeking, a core to drum - drum to core - core to fast disk transfer is queued. If the page is already on the drum, it is left there.

I. Disk to disk transfer

The basic strategy is to create a copy of the information on the drum with a new page status in DHT. Then if the calling process places the page in its PMT & APS and flushes the APS, the automatic mechanisms will cause the disk writes. The detailed algorithm is as follows:

1) Remove request made from request queue. Make DHT entry for new real name. Mark it as a new page without a class code written. Then find the old page.

a) If it is in the CHT, increment the scheduled count and queue a write which will put the drum address in DHT at the new name.

b) If it is on the drum, then queue a read into core for the new real name. Then write it out.

c) If it is on the disk, then cause a disk to core transfer from the old address but putting it into core under the new address. Then write it out.

J. Request New Page

The strategy for the monitor is to place a request in core and place a ptr to it in the right place and alert the swapper. The CPU will then wait until the swapper gives a success or fail return. Success return implies the page is in CHT and the appropriate DHT entry has been made. However, the page has not been cleared. Failure implies that nothing was done. The swapper strategy is to:

1) Look for page in CHT and DHT
   a) If in CHT, generate fail return and set interrupt for error.
   b) If in DHT, same as a).
2) Attempt to remove a page from FCL.
   a) If page removed, make DHT entry and set status as new page. Set WUN bit. Generate success return.
   b) If there is no free core, determine if this process is more important than another process.
      1. If it is, release a page from the least important process and go to 2)a).
      2. If a page cannot be released, generate a fail return.

Miscellaneous Comments

Error Collection

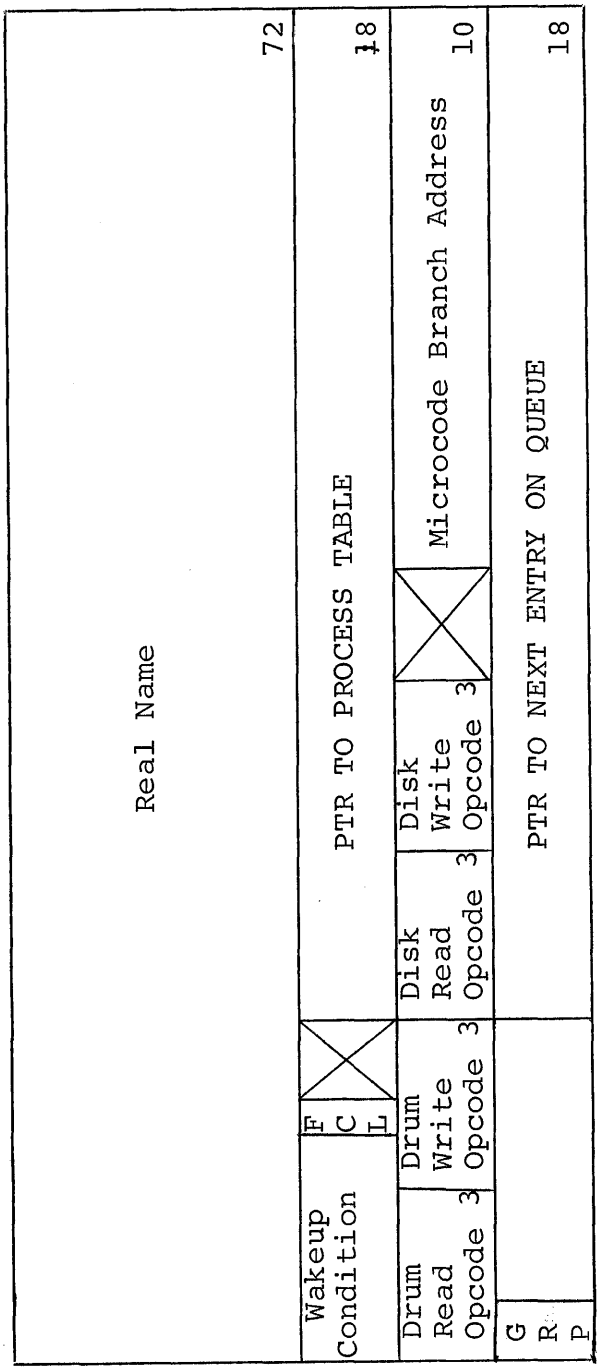
Errors will be made by drum and disk. Data collection on the errors should be made to determine corrective procedures. The following proposal is designed to facilitate collection of such data.

Error information is dumped into a page pointed to by some fixed cells in memory. When the page is half full, the swapper wakes a small process whose goal in life is to save the information in the page on a file. It might, for instance, hash the information into the file and update the counters where necessary. When a process gets awakened, it could make a system call to change the Error Page Pointer (Unique Name).

Diagnostic Routines

These will be specified and described in a document as soon as those desiring these routines make their wishes known to the author.

Request Entry Field Definitions



GRP - This node is a header for a group of requests

Wakeup Condition (CW) -

-any-all
--after
--before

ØØØØ- No wakeup

ØØØ1 Wakeup before the transfer done (after seek completed for disk)

ØØ1Ø Wakeup after the transfer completed

ØØ11 Illegal

Ø1ØØ Illegal

Ø1Ø1 Wakeup before the transfer of last page

Ø11Ø Wakeup after the transfer of last page

Ø111 Illegal

If any-all bit set wakeup generated when any page is transfered. Other bits have meanings described above.

FCL - determines whether page put on end or beginning of Free Core List.

Four opcode fields - these are set by AMC as required by request.

Microcode Branch Address - this field is set by the calling routing to cause a specific cleanup action at the end of the transfer, which usually results in the entry being put in another queue.

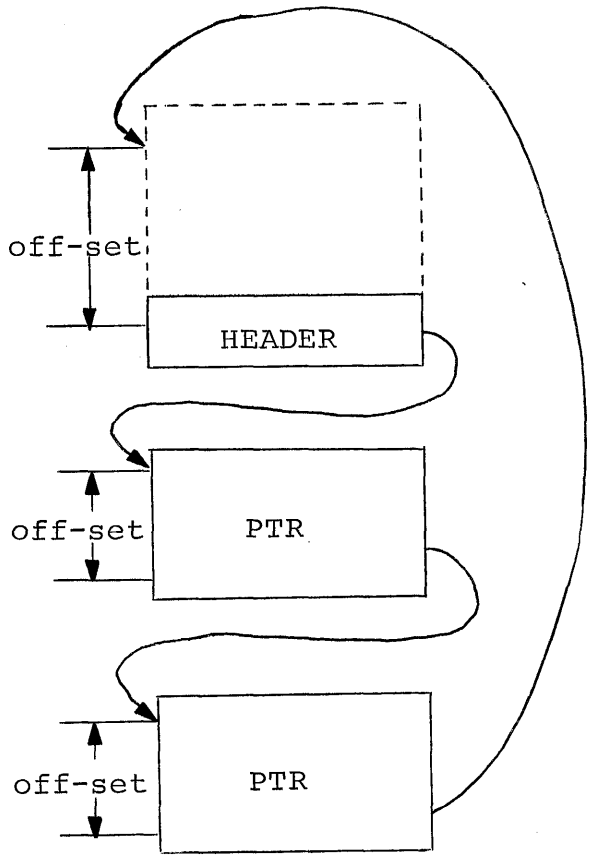
Ptr to next entry on Queue - this field is used to chain the Queue together or to point to the next node in the request.

APPENDIX

List Structures

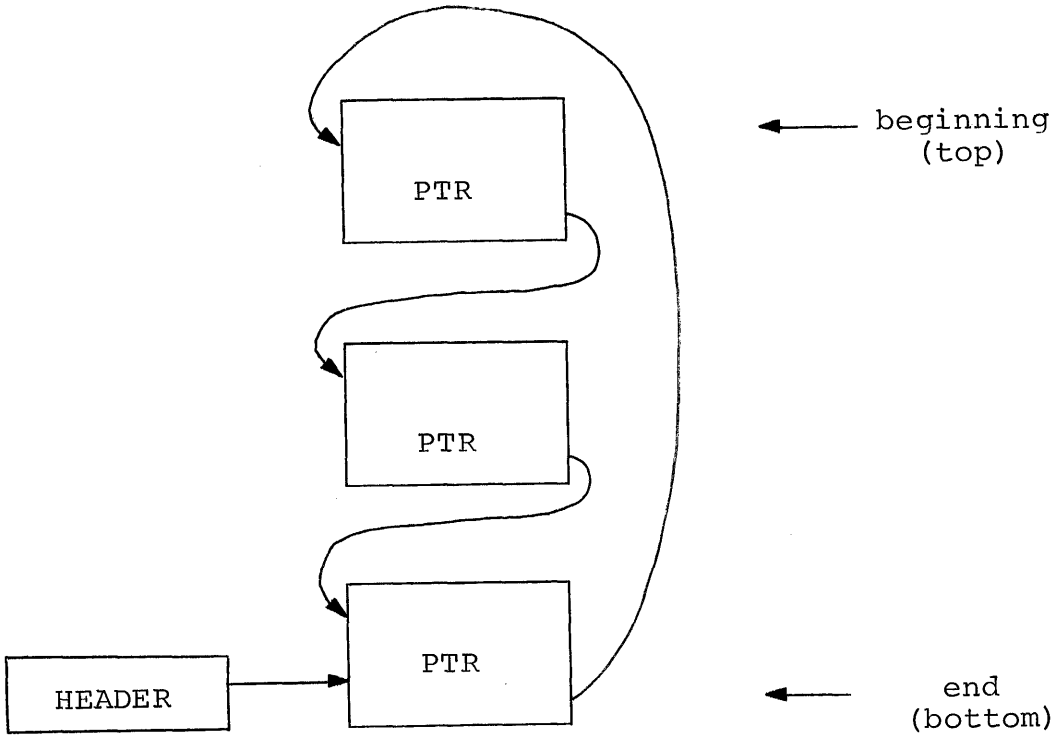
List Type 1

This is a standard offset list (ptr to next element off-set from first word of bead) except that the end pointer points to a pseudo-bead around the head of the list.



List Type 2

This is a circular list structure which can be used as a stack or queue. Elements cannot be removed from end efficiently. Header points to end of list.



List Type 3

This is a list which begins at a "header," (i.e., the header points to the first element on the list) and ends with the element which has a zero in its ptr field.