bcc	title PROCESS MEMORY SYSTEM	prefix/class-number.revision PMS/M-19	
checked	authors Jack Freeman	approval date 1/21/70	revision date
checked		classification Manual	
approved		distribution Company Private	pages 51

ABSTRACT and CONTENTS

This document describes the software part of the Model 5ØØ, Phase I, memory management system. It gives detailed descriptions of the MCALLs on this part of the Monitor.

I Introduction

Model 5ØØ Memory System

The memory system of Model 5ØØ will consist of 256K words of core, 6 million words of drum, and 15Ø million words of disk. The Model 5ØØ System super-imposes a page structure on this storage space. All three levels of storage are sub-divided into 2K-word blocks, called pages. Pages are units of information as well as units of storage space. When we speak of pages of code, pages of data, etc., we mean an amount of code, data, etc., that may be stored in a page of storage. This is just to say that "page" is used in a manner completely analogous to that in which "byte" and "word" are used. When we use "page" to refer to a unit of storage space, we speak of "core pages", "drum pages", and "disk pages" depending on which of the three levels of memory we are referring to. Storage pages have an "origin" as well as an extent (2Ø48 words). Pages of core are 2Ø48 word blocks starting at an address which is congruent to Ø modulo 2Ø48. Similarly, drum and disk pages have fixed "starting addresses" built into the hardware. They are a little different from core pages in that we don't speak of word addresses in connection with these storage devices.

Page Names

Pages of storage have "names," which are functions of their addresses in whichever storage device they reside. Thus the first 2Ø48 words of core (real core locations Ø - 3777_8) are called "core page Ø," the second 2Ø48 words are called "core page 1", and so on. Pages of drum and disk storage space are named in an analogous manner.

Pages of information are also assigned names, called Unique Names. These names are just 48-bit quantities which are assigned to the page when it is created and attached to it throughout its existence in the system. The binding of pages to their Unique Names is built into the storage system's hardware to the extent that whenever a page is written on the drum or disk, its Unique Name is written into a special 48 bit header called the Class Code. There is no hardware analogue of the class code in core storage, but the machinery which transfers pages into and out of core maintains the correspondence between pages and their Unique Names while they are in core. When it reads a page into core from the drum or disk it also reads the page's Unique Name (from the Class Code field on the drum or disk) and saves it in a table in core. Then when the page is written back out the Unique Name is written as the class code of the drum or disk page into which the page is written.

Two tables which reside permanently in core keep track of what information pages are stored on all pages of the two "higher" levels of storage (core and drum). These tables are called, respectively, the Core Hash Table (CHT) and the Drum Hash Table (DHT). There is no analogous record of the contents of disk pages, though consideration has been given to the implementation of a Disk Hash Table in a later system. CHT and DHT are maintained and used by the processor (called the AMC) which transfers pages between the three levels of storage. They are hashed on Unique Name and give, for each page entered in them, the current core/drum page on which the information is stored.

CHT is used in addition by the Map Loaders in the CPUs to find the core page addresses ("names") of information pages to which references are directed by programs running on the CPUs. As this indicates, the use of Unique Names as page addresses is built into the system at a very basic level. In fact, the associative addressing structure provided by these names and the Core and Drum Hash Tables is used in all normal page addressing, even at the level of the basic (software implemented) operating system. Of course, provision is made for bypassing this structure and reading or writing pages specified by their core, drum, or disk address only, but this facility is not used in the normal operation of the system.

Processes

The set of algorithms and data structures which allow the Model 5ØØ to run as a time sharing system are distributed over a number of independent processors and pieces of software. These entities are called collectively the Model 5ØØ Operating System. They conspire together to divide the computing facilities of the system among its users. The fundamental unit of whatnot in terms of which the operating system does its work is called a "process".

A permanent record of the processes belonging to a user of the system is kept in the user's file directory. When a user creates a process (for example, by performing some standard system ENTER procedure) a new entry is made in his file directory. When he destroys the process (by, for example, some sort of LOGOUT procedure), the entry is deleted. Processes which are being actively scheduled and run in the time sharing system are also kept track of in a table called the Process Table (PRT), which is permanently resident in core. Processes with entries in PRT are said to be "active." Those not in PRT (but still recorded in a file directory) are called "dormant" processes. A process may be changed from active to dormant and vice-versa at the explicit request of a user who controls it. In addition the Operating System may make a process dormant when it has exhausted the system resources allocated to it and re-activate it when more are allocated.

Context Blocks

To define a process for the operating system requires a good deal of information. This information is called the "state" of the process. When a process is dormant, its state is defined by its entry in its owner's file directory. Such an entry contains the symbolic name of the process, information for controlling access to the process, and the Unique Name of a special page of the process called its Context Block. This special page contains the information needed to introduce (or re-introduce) the process into the operating system's job stream, that is, to activate the process.

When a process is active, its state is more complex. Some information about it is kept in tables, such as the Process Table and the character I/O line tables, which are resident in core. Information which is needed only when the process is itself in core (or being swapped in or out) is kept in the Context Block page. This page can be thought of as providing temporary storage for the operating system in certain of its functions with respect to the process.

II The Process Memory System

The Process Memory Table

One kind of information the operating system requires about an active process is a list of the Unique Names of all the pages which belong to the process. These names (together with a mapping of the process' address space into them) are needed by the CPU so that it may find the pages to which the process directs references when actually executing instructions. These page names are also required by the Auxiliary Memory Controller (AMC) so that it can identify the pages which it needs to swap into core preparatory to running the process.

The page names, and some additional information about the pages, are kept in a table called the Process Memory Table (PMT) in the Context Block. These tables begin in a standard place (loc. $3ØØ_8$) in each process' Context Block for the convenience of the various parts of the operating system which must reference them. They will initially have room for 128 page names, but later versions of the system may allow for up to 255. That is, the limit of 255 is built into the system in a number of places, but the current 128 page limit is imposed by only the software part of the system.

The operating system software which looks after PMT is called the Process Memory System, which this document is intended to describe. We begin by giving explanations of the contents of entries in PMT. Refer to Figure 1 for a picture of a PMT entry.

UNIQUE NAME:

These two words hold the Unique Name of a page of information. This is the same Unique Name as is written with the page on the disk and drum and kept in the Core Hash Table when the page is in core. It is used by the CPU's map loader when it looks up the page in CHT and by the Swapper when it is swapping the process in or out.

DISK ADDRESS:

This field holds the address at which the disk copy of the page is stored. It is the address which will actually be sent to the disk TSU (Transfer Sub-Unit) when it is required to read the page into core or to write it on the disk. We have to keep such addresses around because there is no provision at the TSU level for addressing pages by their Unique Names. However, the system does not depend on this disk address being correct. When the transfer of a page to or from the disk begins, the contents of the Class Code field of the addressed page is checked for equality with the Unique Name of the page of information it is desired to transfer. If this check fails the transfer is aborted and a "Class Code Error" is reported to the process for which the transfer was being done. A page's Unique Name and Disk Address are called together its "Real Name."

This is as good a time as any to reveal an ugly fact about the Drum Hash Table. First we note that the Core Hash Table is a table entered by hashing the Unique Name of a page and containing for each entry the Real Name of a page and the absolute address of the core page in which the page is currently stored. Ideally the Drum Hash Table would be completely analogous and each entry would contain a Real Name and a drum page address of the current drum copy of the page. This implementation was not possible, simply because of the amount of core storage which such a table would require. Instead, DHT entries contain only the Disk Address word of the Real Name. Except for the loss in elegance this seldom causes any problems. It just means that in certain cases we have to do an otherwise unecessary read from the drum to compare a Class Code with a Unique Name.

So, the Disk Address word in PMT entries is used to find the page whose Unique Name appears in the entry both on the disk and on the drum, but in neither case is it considered the final authority in the matter since we always make the comparison between Class Code and Unique Name.

ACCESS LOCK:

CONTROL LOCK:

The operating system provides for sub-dividing processes among up to 8 separate programs. This part of the system is entirely software and is described in the document MISPS/M-7. Programs coexisting in a process are called sub-processes of the process. In order that this sub-division be useful it is necessary that sub-processes be able to protect themselves from other sub-processes. In particular, they must be able to protect their memory from accidental or malicious access by programs which they don't trust. For example, a debugger cannot in general hope to be a success if it is freely accessible by the programs it is trying to debug. Since the process' and therefore the sub-processes' memory is represented by PMT entires, this means that we must have some way of allowing sub-processes to control access to PMT entries. This is implemented as follows. Each sub-process has a KEY and a NAME. When a sub-process acquires a PMT entry, its NAME is put into the entry's

ACCESS and CONTROL LOCKs. Then whenever a sub-process requests that some operation be performed on the PMT entry, its KEY is compared with one of these LOCKs. If the two fields have no bits in common, the operation is not allowed. Most operations on PMT entries require a KEY which fits the entry's CONTROL LOCK. The one operation of putting the entry into a map requires only a KEY which fits the ACCESS LOCK.

FP (FILE PAGE):

With a few exceptions, all pages in the system are either pages of files or pages of "private memory." These two kinds of pages are quite different. Private memory pages have no existence outside of the process to which they belong, in the sense that there is no way by which other processes may get at them or even find out that they exist. File pages on the other hand are recorded (that is, their Unique Names are recorded) in the File Directories of the user's to whom they belong. Access to them is carefully controlled by the Basic File System's protection mechanism and as a result they can be shared between processes. The Basic File System contains an MCALL by which a sub-process can put the Unique Name of a file page into a PMT entry in its process. When this is done, the entry's FP flag is set for the convenience of other MCALLs (see below) which must be able to distinguish between pages which are pages of files and pages which are private memory pages. Note: The use of the first 2 bits of the Unique Name to indicate page type makes the FP flag redundant for this purpose, but it is still required because of the existence of the privileged MCALL which allows an arbitrary Real Name to be put into PMT.

NC (NO CHARGE):

The operating system limits the number of pages which a process may have on the drum at one time. Basically, the process is charged for every page that it has in its Drum Working Set (discussed below). In certain cases many processes will have common pages, such as the code pages of the BASIC and FORTRAN systems, in their Drum Working Sets, and in these cases we don't want to charge each process for the use of these pages. We avoid this by setting the NC bit in PMT when the process places such a page. Pages marked with NC will not be counted in computing the size of the process' Drum Working Set.

RO (READ ONLY):

When a process places file pages in PMT, the Basic File System sets the read-only status obtained when the file was opened into this bit in the PMT entries. This is to insure that the protection on the page provided by the file system is not relaxed when the page appears in PMT. The RO bit in PMT is actually used by the CPU to trap stores into the page.

RF (REFERENCE FLAG):

The system trys to make sure that the pages in the Core and Drum Working Sets of a process are the ones that the process is referencing most frequently. In order to do this, it must somehow be kept informed as to what pages the process is referencing. The CPU's Map Loader provides this information by setting the RF flag in PMT whenever it loads the corresponding page into its map.

SF (SCHEDULED FLAG):

When a program causes the Map Loader to load a page into the CPU's map, the Map Loader looks the page up in the Core Hash Table using the Unique Name in the appropriate PMT entry. Now it is possible that the page is in core for some other process but not supposed to be available to the process in which the program is running. Giving the program access to the page under these conditions will in general lead to chaos, since the core storage management system depends on knowing how many processes have access to the pages in core. The SF bit is used to prevent this illegal access. It is set by the core management system if the process is authorized to access the page, and the CPU will trap if it is asked to load a page with SF = Ø into its map.

CCE (CLASS CODE ERROR):

When the pages of a process' Core Working Set are being read into core the Unique Names in PMT are compared with the Class Codes on the pages read. If the comparison fails, the read is aborted and the CCE flag in the PMT entry is set. The SF bit is of course reset. If the process tries to reference the page it will get a trap from the CPU, at which time CCE can be tested to determine the source of the problem.

This completes the description of PMT itself. Before we go on to describe the MCALLs with which a program can do things to PMT we must describe two important sets of pointers into the table. These are the Process Map (PRMAP), used by the CPU's Map Loader, and the Core Working Set, (CWS) used by the Swapper.

The Active Page Table

When it is time to bring a process into core so that it may execute instructions on a CPU, a request is sent to the Swapper to bring in the pages the process needs in order to run. The Swapper is given a pointer to the process' entry in PRT. In the PRT entry the Swapper finds the Real Name of the process' Context Block. It brings this page into core. In the Context Block is a table, called the Active Page Table (APT), which contains pointers into the Process Memory Table. Entries in APT are marked as to whether the pages they point to are to be swapped in or not. The set of pages which are marked to be swapped in is called the Core Working Set (CWS) of the process. The Swapper scans APT and reads all CWS pages into core. When these reads are completed the process is said to be loaded and is available to be run on a CPU.

Figure 2 gives the format of an entry in APT. We now explain the various fields shown in the figure.

USE HISTORY:

This field is used by the system to keep a history of references the process directs to the page the entry points to. It is updated periodically from the RF flag in PMT and used by the routines (described below) which maintain the Core Working Set.

PAGE LOCK:

It is possible to lock pages into core, that is , to exempt them from the algorithms which cause dirty pages to be written back on the drum and pages not in any Core Working Set to be released from core. The operating system can lock pages directly by turning on bits in the pages' entries in the Core Hash Table. Certain privileged User Programs will also need to insure that pages are kept in core. The Monitor will provide an MCALL which can be used to do this. When a process executes this MCALL, the PAGE LOCK field of its CWS entry for the page will be set to a code identifying the lock bit in CHT for which the process is responsible. Details will be supplied at a later date.

KEEP:

LOCK:

These fields are intended to allow a program to designate elements of its Core Working Set as more important than others. No operations on them will be implemented in this current version of the Process Memory System, however.

DWS:

In addition to the Core Working Set there is another subset of APT called the Drum Working Set. It is the set of pages which are being kept on the drum for the process. It is a super-set of the Core Working Set and is maintained entirely by the software parts of the operating system. The DWS bit in an APT entry is set if the page pointed to is in the process' Drum Working Set. The Drum Working Set is called DWS for short.

CWS:

This is the bit the Swapper uses to determine whether an APT entry points to a page to be swapped in. It is set if the page is to be swapped in (i.e., is in the process' Core Working Set) and reset if it isn't.

PMT INDEX:

This is an index into the Process Memory Table and points to a Real Name of a page.

The Process Map

At location $2\emptyset\emptyset_8$ in each process' Context Block is stored the process' Process Map (PRMAP). This is in the form of 128 12-bit bytes and is shown in Figure 3. Each byte of PRMAP contains two pieces of information.

RO:

This bit being set marks the page as Read Only for the process. Its value is loaded into the CPU's physical map by the Map Loader when the process makes its first reference to the page in its address space to which the byte corresponds.

PMT INDEX:

This index into PMT is used by the Map Loaders in the CPU's to find a Unique Name with which to hash into CHT.

There are no MCALLs by which a program may directly read or write the Process Map. It effectively refers to PRMAP, however, when it modifies its sub-process map by the operations described in the document on the Sub-process System (MISPS/M-7).

III The MCALLs

The following pages describe the MCALLs by which programs can explicitly modify and examine the memory system of the process in chich they are running. Most of the calls are for performing operations on the process' PMT and APT.

A PMT entry is free if its CONTROL LOCK is zero.

A PMT entry is empty if its DISK ADDRESS is zero.

A sub-process controls a PMT entry if the bit-wise AND of the sub-process' KEY with the CONTROL LOCK of the entry is non-zero

A sub-process has access to a PMT entry if the bit-wise AND of its KEY with the ACCESS LOCK of the entry is non-zero, or if it controls the entry.

NPMTE is the maximum number of entries which a Process Memory Table can contain. The value of this parameter is 128.

Associated with a process' Core Working Set are three "lengths":

LCWS - the number of pages currently in the Core Working Set, OLCWS - the value of LCWS at which "Core Working Set Overflow" will occur,

MLCWS - the maximum value to which OLCWS may be set.

LCWS is maintained by the Process Memory System, being incremented when a page is added to CWS and decremented when one is removed. OLCWS can be set by one of the MCALLs to any value between LCWS and MLCWS. MLCWS will be a parameter with value 32 in Phase I.

Three exactly analogous lengths are associated with each process' Drum Working Set. They are called LDWS, OLDWS, and MLDWS. The Phase I value for MLDWS is 64.

A process Core Working Set is <u>full</u> if LCWS = OLCWS. Its Drum Working set is full if LDWS = OLDWS.

ACQPMT - Acquire and Initialize a PMT Entry

Declaration:

FUNCTION ACQPMT(PMTX), FRETURN, MONITOR ← 5Ø;

Success Return:

RETURN PMTX;

Failure Returns:

(1) FRETURN('PMI', 11Ø) unless

(a) $1 \leq$ PMTX $\leq$ NPMTE, or

(b) PMTX = -1

(2) FRETURN('PMO', 118) if PMTX = -1 and there are no free PMT entries.

(3) FRETURN('PMA', 119) if PMTX ≠ -1 and PMT[PMTX] is not free.

Action:

If PMTX = -1, PMT is searched for a free entry and the index of the first one found is assigned to PMTX. PMT[PMTX] is cleared and then its CONTROL LOCK and ACCESS LOCK fields are set to the NAME of the calling sub-process. PMTX is returned as the MCALL's value.

NPPMT - Create a Private Memory Page and Put its Real Name into PMT

Declaration:

FUNCTION NPPMT(PMTX), FRETURN, MONITOR ← 51;

Success Return:

RETURN;

Failure Returns:

(1) FRETURN('PMI', 11Ø) unless 1 ≤ PMTX ≤ NPMTE.

(2) FRETURN('PMC', 121) unless PMT[PMTX] is controlled by the calling sub-process.

(3) FRETURN('PMF', 122) unless PMT[PMTX] is empty.

(4) FRETURN('DWF', 123) if the process' Drum Working Set is full and the calling sub-process does not have Default DWS Overflow (DDWSO) selected in its STATUS BIT word.

(5) FRETURN('CWF', 124) if the process' Core Working Set is full and the calling sub-process does not have Default CWS Overflow (DCWSO) selected in its STATUS BIT word.

(6) FRETURN('KSE', 125) if the process' disk space is exhausted.

Action:

A private memory page is created and its Unique Name and Disk Address are put into the appropriate fields of PMT[PMTX]. The page is put into the Core and Drum Working Sets of the process. The SF bit in PMT[PMTX] is

NPPMT (continued)

set and the other status bits of the entry are cleared.

RNPMT - Put Specified Real Name into a PMT Entry

Declaration:

```
FUNCTION RNPMT(PMTX, Long UN, Integer DKA), FRETURN,
                                         MONITOR ← 52;
```

Success Return:

RETURN;

Failure Returns:

(1) FRETURN('STS', 127) unless the calling subprocess has the privileged System Diagnostic status (SD).

(2) FRETURN('PMI', 11Ø) unless 1 ≤ PMTX ≤ NPMTE.

(3) FRETURN('PMC', 121) unless the calling sub-process controls PMT[PMTX].

(4) FRETURN('PMF', 122) unless PMT[PMTX] is empty.

Action:

The Unique Name, UN, and Disk Address, DKA, are simply copied into the appropriate fields of PMT[PMTX]. The status bit FP is set and the other status bits are cleared.

CLRPMT- Release Page from PMT Entry

Declaration:

FUNCTION CLRPMT(PMTX), FRETURN, MONITOR ← 53;

Success Return:

RETURN;

Failure Returns:

(1) FRETURN('PMI', 11Ø) unless 1 ≤ PMTX ≤ NPMTE.

(2) FRETURN('PMC', 121) unless the calling sub-process controls PMT[PMTX].

Action:

The page whose Real Name appears in PMT[PMTX] is released from the Core and Drum Working Sets of the process. If the page is a private memory page, (i.e., if FP = Ø) it is also "released" from the disk, with the effect that it ceases to exist in the system. The PMT entry is cleared, with the exception of the CONTROL and ACCESS LOCK fields, which are not changed.

DELPMT - Release PMT Entry

Declaration:

```
   FUNCTION DELPMT(PMTX), FRETURN, MONITOR← 54;
```

Success Return:

```
   RETURN;
```

Failure Returns:

(1) FRETURN('PMI', 11∅) unless 1≤ PMTX≤ NPMTE.

(2) FRETURN('PMC', 121) unless the calling sub-process controls PMT[PMTX].

Action:

The MCALL does exactly what CLRPMT does -- releases the page from the process' CWS and DWS and destroys it if it is a private memory page -- and in addition frees the PMT entry by clearing its ACCESS LOCK and CONTROL LOCK. All pointers to PMT[PMTX], from sub-process maps and the process map, are deleted at this time.

SPMTAL - Set the ACCESS LOCK of a PMT Entry

Declaration:

FUNCTION SPMTAL(PMTX, AL), FRETURN, MONITOR←55;

Success Return:

RETURN;

Failure Returns:

(1) FRETURN('PMI', 11∅) unless 1≤ PMTX ≤NPMTE.

(2) FRETURN('PMC', 121) unless

(a) the calling sub-process controls PMT[PMTX], or

(b) the calling sub-process has access to PMT[PMTX] and the exclusive or of AL with the ACCESS LOCK of PMT[PMTX] contains no bits which are not set in the calling sub-process' KEY.

Action:

AL is set into the ACCESS LOCK field of PMT[PMTX].

SPMTCL - Set the CONTROL LOCK of an SPT Entry

Declaration:

```
   FUNCTION SPMTCL(PMTX, CL), FRETURN, MONITOR←56;
```

Success Return:

```
   RETURN;
```

Failure Returns:

(1) FRETURN('PMI', 11Ø) unless 1≤ PMTX≤NPMTE.

(2) FRETURN('PMC', 121) unless the calling sub-process controls PMT[PMTX].

(3) FRETURN('SPC', 134) if the exclusive or of CL with the CONTROL LOCK of PMT[PMTX] contains any bits which are not set in the KEY of the calling sub-process.

Action:

CL is set into the CONTROL LOCK field of PMT[PMTX]. If CL is Ø, the PMT entry is released with DELPMT.

SPMTRO - Set the Read Only Bit in a PMT Entry

Declaration:

FUNCTION SPMTRO(PMTX, RO), FRETURN, MONITOR←57;

Success Return:

RETURN;

Failure Returns:

(1) FRETURN('PMI', 11Ø) unless 1≤ PMTX≤NPMTE.

(2) FRETURN('PMC', 121) unless the calling subprocess controls PMT[PMTX].

(3) FRETURN('FPR', 135) if FP and RO are set in PMT[PMTX] and the offered RO is Ø.

Action:

The value of RO is copied into the PMT entry's RO field.

READPMT - Read a PMT Entry

Declaration:

```
   FUNCTION READPMT(PMTX, ARRAY PMTE), FRETURN, MONITOR←58;
```

Success Return:

```
   RETURN;
```

Failure Returns:

(1) FRETURN('PMI', 11Ø) unless 1≤PMTX≤NPMTE.

Action:

Five words are copied into the caller's array. The first four are the four words of PMT[PMTX]. The fifth is the APT entry which points to PMT[PMTX] if there is such an entry, or Ø if there isn't.

PPDWS - Put Page in Drum Working Set

Declaration:

```
FUNCTION PPDWS(PMTX), FRETURN, MONITOR←65;
```

Success Return:

```
RETURN;
```

Failure Returns:

(1) FRETURN('PMI', 11Ø) unless 1≤PMTX≤NPMTE.

(2) FRETURN('PMC', 121) unless the calling sub-process controls PMT[PMTX] or has access to it.

(3) FRETURN('PME', 128) if PMT[PMTX] is empty.

(4) FRETURN('DWF', 134) if the process' Drum Working set is full and the calling sub-process doesn't have DDWSO (Default action on Drum Working Set Overflow) set in its SPT entry.

Action:

The page whose Real Name appears in PMT[PMTX] is transferred from the disk to the drum. PMTX is entered in the process' Active Page Table and the APT entry thus created is initialized according to

```
UH← -1
DWS← 1
PMT← PMTX
```

If the page is already in the process' Drum Working Set no action is taken.

PPDWS - Continued

If the process' Drum Working Set is full and the calling sub-process has DDWSO set in its SPT entry, the system will delete some page from DWS in order to make room for the new entry. The algorithm the system uses to choose the page to be deleted is described elsewhere in this document.

PPCWS - Put Page in Core Working Set

Declaration:

FUNCTION PPCWS(PMTX), FRETURN, MONITOR ← 66;

Success Return:

RETURN;

Failure Returns:

(1) FRETURN('PMI', 11Ø) unless $1 \leq \text{PMTX} \leq \text{NPMTE}$;

(2) FRETURN('PMC', 121) unless the calling sub-process controls PMT[PMTX] or has access to it.

(3) FRETURN('PME', 128) if PMT [PMTX] is empty.

(4) FRETURN('DWF', 134) if the process' Drum Working Set is full and the calling sub-process doesn't have DDWSO (Default action on Drum Working Set Overflow) set in its SPT Entry.

(5) FRETURN('CWF', 132) if the process' Core Working Set is full and the calling sub-process doesn't have DCWSO (Default action on Core Working Set Overflow) set in its SPT Entry.

Action:

The page named by PMT[PMTX] is transferred to the drum, if necessary, using PPDWS. The CWS bit in the resulting APT entry is then set, with the effect that the next time the process is read in by the AMC the page will be swapped in as part of it. Note that this

PPCWS - Continued

means that PPCWS does not cause the page to become available "immediately". A page fault will still occur if the page is referenced before the next time the process is swapped in.

If the process' Core Working Set is full, some page will be removed from it to make room for the new page.

DPDWS - Delete Page from Drum Working Set

Declaration:

FUNCTION DPDWS(PMTX), FRETURN, MONITOR ← 67;

Success Return:

RETURN PMTX'; where PMTX' will be the PMT index of the page deleted or -1 if the page named by PMT[PMTX] was not in DWS.

Failure Returns:

(1) FRETURN('PMI', 11Ø) unless

(a) 1 $\leq$ PMTX $\leq$ NPMTE, or

(b) PMTX = -1.

(2) FRETURN('PMC', 121) if PMTX ≠ -1 and the calling sub-process neither controls nor has access to PMT[PMTX].

(3) FRETURN('DWE', 159) if PMTX = -1 and the process' Drum Working Set is empty (which is very unlikely).

Action:

The page named by PMT[PMTX] is released from the process' Core and Drum Working Sets. Its entry in APT is deleted and the SF bit in PMT[PMTX] is cleared.

If the calling sub-process supplies -1 as the value of PMTX, the system chooses a page to delete according to the same rules used to correct the Drum Working Set Overflow condition.

DPCWS - Delete Page from Core Working Set

Declaration:

FUNCTION DPCWS(PMTX), FRETURN, MONITOR ← 68;

Success Return:

RETURN PMTX'; where PMTX' will be the PMT index of the page deleted or -1 if the page named by PMT[PMTX] wasn't in CWS.

Failure Returns:

(1) FRETURN('PMI', 11Ø) unless

(a) $1 \leq$ PMTX $\leq$ NPMTE, or

(b) PMTX = -1.

(2) FRETURN('PMC', 121) if PMTX ≠ -1 and the calling sub-process neither controls nor has access to PMT[PMTX].

(3) FRETURN('CWE', 158) if PMTX = -1 and the process' Core Working Set is empty.

Action:

The page whose Real Name appears in PMT[PMTX] is released from the Core Working Set of the process. The CWS bit in its APT entry and the SF bit in the PMT entry are reset. The page becomes unavailable to the process immediately.

If PMTX = -1, the system chooses a page to delete, using its DCWSO algorithm.

READ'LWS - Read Length of Working Set

Declaration:

FUNCTION READ'LWS(CODE),FRETURN, MONITOR←7Ø;

Success Return:

RETURN LNGTH; where LNGTH is that one of 6 working set "lengths" selected by CODE as described below.

Failure Return:

(1) FRETURN('ARG', 191) unless Code has an acceptable value. (See Below).

Action:

Code is used to select one of 6 possible working set length according to the scheme given below. The selected length is the value of the MCALL.

CODE	Length
Ø or 'CWS'	LCWS
1 or 'OCW'	OLCWS
2 or 'MCW'	MLCWS
3 or 'DWS'	LDWS
4 or 'ODW'	OLDWS
5 or 'MDW'	MLDWS

SET'LWS - Set Overflow Length of Working Set

Declaration:

```
     FUNCTION SET'LWS(CODE,LNGTH),FRETURN,MONITOR←71;
```

Success Return:

```
     RETURN;
```

Failure Returns:

(1) FRETURN('ARG', 191) unless CODE has one of the 4 values 1, 4, 'OCW', 'ODW'.

(2) FRETURN('WSL', 192) if

(a) CODE = 1 or 'OCW', and LNGTH doesn't satisfy LCWS $\leq$ LNGTH $\leq$ MLCWS, or

(b) CODE = 4 or 'ODW', and LNGTH doesn't satisfy LDWS $\leq$ LNGTH $\leq$ MLDWS.

Action:

If the value of CODE is 1 or 'OCW', OLCWS is set to LNGTH. If the value of CODE is 4 or 'ODW', OLDWS is set to LNGTH.

IV System Maintenance of Core Working Sets

A process that knows what it's doing can use the above operations to insure that its Core Working Set contains the pages it is currently referencing and no others. Not all processes will be clever enough or industrious enough to do this, however, so the basic system will incorporate procedures for automatically maintaining Core Working Sets in a reasonable state. The application of these procedures to a process' CWS can be controlled by the process' currently active subprocess through setting and re-setting the DCWSO bit in the sub-process' STATUS BIT word.

If CWS maintenance is left entirely to the basic system it will be handled as follows:

(1) Pages will be added to CWS when a CPU reference generates a Page Not in Core (PNIC) trap, i.e., when page faults occur.

(2) A use history will be kept for each page which appears in CWS.

(3) When the use history of a page in CWS indicates that the page is no longer being used by the process, the page will be removed from CWS.

(4) If CWS is full when a page fault occurs, the use histories will be used to select a current entry in CWS for deletion.

A. Use Histories

Use histories for the members of CWS are kept in the 8-bit UH fields of CWS entries. These fields tell us about references to pages during the last 8 times the process ran on the CPU. The left most bit (bit Ø) records references during the most recent time, bit 1 records those during the next most recent, and so on. Figure 2 gives a sample CWS entry with an interpretation of its USE HISTORY field.

When a page is first put in CWS its UH field is initialized to all ones. The effect of this is to assure new entries preferential treatment by the algorithm which chooses an entry to delete when a CWS overflow occurs. The UH fields are updated periodically as follows:

(1) Each UH field is shifted 1 bit right, the contents of the right most bit (i.e., the most ancient historical information) being discarded.

(2) The RF (Reference Flag) bits in the PMT entries pointed to by the CWS entries are copied into the now vacant left most bit

positions of the UH fields. The RF bits in PMT are reset after they are recorded in UH.

Steps (1) and (2) are not perfomed on CWS entries which point to PMT entries with SF = Ø. The pages pointed to by such entries are not even in core (as far as the process in concerned) and it would be misleading to note that they had not been referenced. A process which adds pages to its Core Working Set in anticipation of its need for them will not infrequently have such entries in CWS when it is dismissed, since pages are brought in not when they are added to CWS but the next time the Swapper reads the process in.

The record-keeping operations just outlined are performed by the system regardless of the state of the CWS maintenance strategy flag, DCWSO. Setting or resetting DCWSO enables or disables the system machinery for automatically removing "unused" pages from CWS and for automatically correcting the CWS overflow condition.

B. Automatic Deletion of Pages From CWS

This gets invoked immediately after all CWS USE HISTORY fields have been brought up to date preparatory to dismissing the process. It scans CWS, looking for entries with the first (i.e., left-most) N bits clear, where N is an as yet unspecified integer between 1 and 8. An entry which meets this condition is deleted from CWS if its KEEP and LOCK fields are both clear. This qualification, KEEP = LOCK = Ø, allows a process to use the system's maintenance strategy in general but except special pages from it.

C. Correction of CWS Overflow

Overflow of the Core Working Set occurs when a process attempts to add a new page to CWS and CWS already contains the maximum number of entries it is allowed to hold. If DCWSO is set when this happens, the system will choose some current member of CWS and delete it to make room for the new page.

The system will consider for deletion the elements of CWS which have been referenced least recently. It uses the USE HISTORY fields to identify these elements, first extending the fields by pre-fixing to them the RF flags from the PMT entries with which they correspond. It scans these extended use histories for a maximally long string of leading zeroes. If two or more entries have this same maximum number, say M, of leading zeroes it attempts to differentiate them by looking for maximally long sequences of zeroes starting at bit position M+2. It will continue this until either a single entry is isolated or UH is exhausted. In the latter case the first entry encountered in the final scan will be deleted. This sounds complicated, but it is fortunately exactly equivalent to selecting the first entry in CWS whose extended UH field, considered as a number, is minimal.

CWS entries with LOCK = 1 will not be deleted. An entry with KEEP = 1 will be deleted only if there are no entries with KEEP = LOCK = ∅. It should never happen that the entire CWS is LOCKed.

The procedure just described will always be used to correct a CWS overflow which occurs in Monitor Mode, regardless of the setting DCWSO. This is necessary since it would not be possible in general to continue a Monitor function after giving control to the user to handle the overflow. For the same reason PNIC traps which occur in Monitor Mode will not be sent to the user.

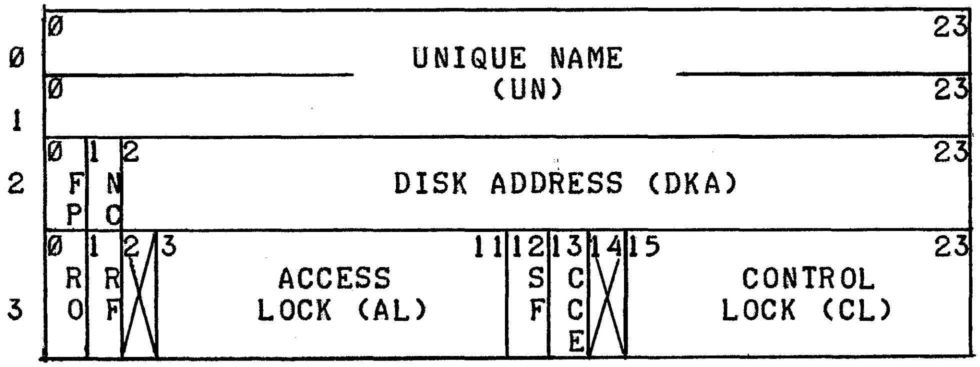
V System Handling of Page-Not-In-Map Traps

When a process makes a reference to a page for which no PMT index appears in the process' PRMAP, a Page-Not-In-Map (PNIM) trap occurs. The action taken by the operating system in such a case depends on the value of the flag DPNIM in the Sub-Process Table entry for the currently running program. If DPNIM is reset, the system will send a trap to the program. That is, it will call the program at a special "trap" entry point, passing it the information that a PNIM trap has occurred and the address to which the trapped reference was made.

If DPNIM is set, the system's "Default PNIM Strategy" is invoked. This strategy is to create a new page for the program just as if a call on NPPMT had been made, put the PMT index returned by NPPMT into the empty byte in PRMAP, and return control to the trapped instruction.

PNIM traps which occur while the program is executing in the Monitor are always handled with the default strategy.

Format of an Entry in a Process Memory Table



FP - File Page

NC - No Charge

RO - Read-Only

RF - Reference Flag

SF - Scheduled Flag

CCE - Class Code Error

Figure 1

Format of an Entry in an Active Page Table

	Ø 7	8 11	12	13	14	15	16 23
Ø	USE HISTORY (UH)	PAGE LOCK (PGL)	DWS	CWS	KEEP	LOCK	PMT INDEX (PMT)

An Example of an APT Entry

Ø	1	2	3	4	5	6	7	8	9	1Ø	11	12	13	14	15	16	17	18	19	2Ø	21	22	23
1	1	Ø	1	Ø	Ø	1	1	Ø	Ø	Ø	Ø	1	1	Ø	Ø	Ø	1	Ø	Ø	Ø	1	Ø	1

The entry tells us that the page whose Real Name appears in PMT[69] is in the process' Drum and Core Working Sets. The page is not locked in core for this process. Nor is it KEPT or LOCKed in the working sets. The process has made references to the page during

the last interval,

the last interval but 1,

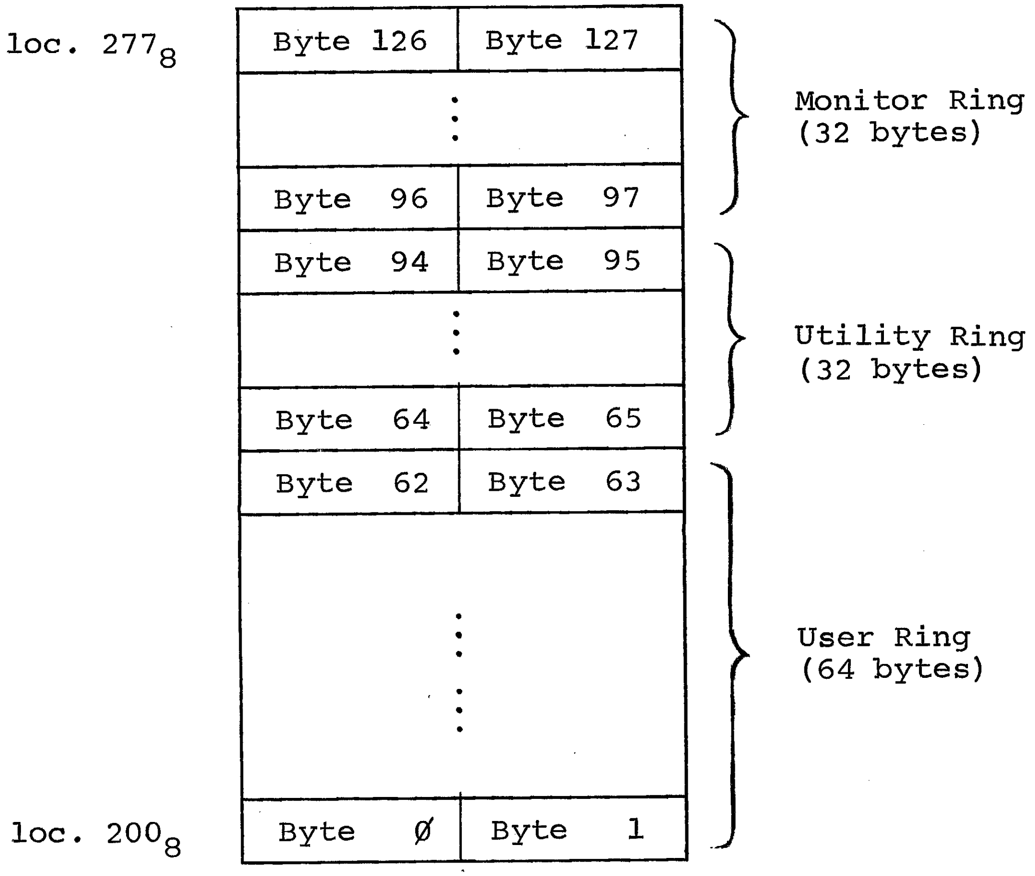
the last interval but 3,

the last interval but 6, and

the last interval but 7.

Figure 2

The Process Map in a Context Block



Format of a Byte in PRMAP

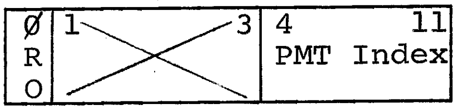


Figure 3