bcc	title	prefix/class-number.revision
	THE CORE WORKING SET	CWS/W-5

checked	authors	approval date / revision date	
Raymie Schulz	Jack Greene	classification: Specification	
checked: Larry Barnes			
approved: Mel		distribution: Company Private	pages: 11

ABSTRACT and CONTENTS

This document describes the Core Working Set (CWS) and specifies the basic system routines which will be provided for handling it.

TABLE OF CONTENTS

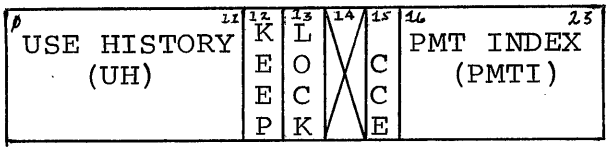
THE CORE WORKING SET

The Core Working Set of a process is the set of pages which must be brought into core before the process will be allowed to run. Ideally, the Core Working Set will consist of precisely those pages which the process references during its time on the CPU. References to pages not in CWS and therefore not in core are Page Faults and generally cause the process to be dismissed prematurely, with the implication that some of the resources expended in swapping it in were wasted. On the other hand, resources spent in bringing in pages which don't get referenced are just as clearly wasted.

I REPRESENTATION OF THE CORE WORKING SET

A process' Core Working Set, exclusive of the resident system tables and code, is kept track of in a table, also called the Core Working Set (CWS), in its Context Block. This table can contain up to 48 entries but the number of entries which are actually available to a process will depend on the user to whom the process belongs. One of the parameters in each user's User Profile is the Maximum Length of the Core Working Set (MLCWS).

An entry in CWS is a single word with the format



The first 3 fields are for the use of the system's Core Working Set Maintenance routines and will be described later. The PMT INDEX field points to an entry in the Process Memory Table and is used by the Swapper to find the pages to be read in when it is bringing the process into core and to identify the pages to be released when the process is written out. The CCE (Class Code Error) bit is set by the Swapper if it finds that the Unique Name in PMT is not the same as the Class Code on the page addressed by the Disk Address in PMT.

OPERATIONS ON CWS

The following operations on CWS may be performed through calls on the basic system. They are described as if the caller were sub-process S. Most of them are legal only if S has Modify CWS (MCWS) status.

(1) Put PMT index N into CWS

This call requires that

(a) S has MCWS status

(b) PMT(N) is not free (CL(N) $\neq \emptyset$)

(c) N is in the interval [M,NPMTE], where M is 1 if S has Master Sub-Process (MSP) status, and NMPMTE+1 otherwise.

The system makes a new entry in CWS, putting N in the PMT INDEX field and initializing the USE HISTORY, KEEP, LOCK and CCE fields. (All bits in USE HISTORY are turned on and KEEP, LOCK, and CCE are cleared.)

If N already appears in CWS a duplicate entry is not made. The USE HISTORY, KEEP, LOCK and CCE fields of the existing entry are initialized.

It may happen that CWS is full (i.e., LCWS = MLCWS, where LCWS is the number of entries currently in CWS and MLCWS is the maximum number allowed). The action taken in this case depends on whether S is using the system's CWS maintenance strategy or is handling its own maintenance. In the former case some PMT index currently in CWS is thrown out and replaced with N. In the latter, a Core Working Set Overflow

(CWSO) trap is sent to S.

(2) Delete PMT index N from CWS

This call fails unless

(a) S has MCWS status

(b) N is in the interval [M, NPMTE], where M is defined as for operation (1).

CWS is searched for an entry with N in its PMT INDEX field. If one is found it is removed from CWS and the function returns with N as its value. If N is not in CWS, the function still succeeds but returns -1 instead of N.

(3) Delete an entry from CWS

This function takes no arguments and requires only that

(a) S has MCWS status

(b) there is at least one CWS entry with LOCK = Ø.

The system selects some CWS entry (with PMT INDEX pointing to the sub-process part of PMT) by the same algorithm it uses to handle CWS overflow, and deletes it. The PMT INDEX is returned as the value of the call.

(4) Set CWS Maintenance Strategy to M

This is legal iff

(a) S has MCWS status

(b) S controls itself (NAME(S) $\wedge$ KEY(S) $\neq \emptyset$)

M (which must be $\emptyset$ or 1) is copied into the Default Maintenance of Core Working Set (DMWS) bit in the Status Bit field of S's entry in the Sub-process Table. The effectoof this is to disable or enable operation of the system's CWS maintenance routines, which are described in the next section.

(5) Set CWS KEEP field for PMT index N to K

This requires that

(a) S has MCWS status

(b) N is in the interval [M,NPMTE], where M is defined as for operation (1).

(c) N is in CWS.

K ($\emptyset$ or 1) is set into the KEEP field of the CWS entry which has N in its PMT INDEX field. The purpose of the KEEP field is to allow a process to protect specific CWS entries from being deleted by the system's CWS maintenance routines. Entries with KEEP set will be deleted only as a last resort.

(6) Set CWS LOCK field for PMT index N to L

The LOCK field is like the KEEP field, but "stronger" - the system's maintenance routines will not delete a LOCKed entry from CWS.

Writing LOCK is allowed iff

(a) S has MCWS status

(b) N is in the interval [M,NPMTE], where M is defined as for operation (1).

(c) N is in CWS

(d) either CWS is not full (LCWS$<$MLCWS) or there is at least one CWS entry with PMTI $\neq$ N, which has LOCK = Ø. This condition helps insure that a full CWS will not be completely LOCKed. LOCK is set to L (Ø or 1).

(7) Read CWS entry M

This is a completely un-privileged operation and only requires that

(a) Ø$\leq$M$\leq$47.

It returns the contents of the Mth CWS entry.

(8) Read LCWS

This function returns LCWS, the number of entries currently in CWS. There are no restrictions on calling it.

(9) Read MLCWS

This is another unrestricted call, which returns the maximum number of entries which CWS is allowed to contain. The value of MLCWS is read from the Context Block, not from the User Profile. The Context Block value of MLCWS may be different from the User Profile one, as a result of a call on the function next to be described.

(10) Set MLCWS to N

Setting MLCWS is a highly privileged operation. It requires

(a) S has MCWS status

(b) S has Executive (EX) status

(c) $LCWS \leq N \leq 47$

(d) either $LCWS < N$ or there is at least one CWS entry with LOCK = Ø.

The Context Block copy of MLCWS is set to N. It is assumed that the caller of this function has determined that N is no greater than the User Profile value for MLCWS.

III 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 turned on and off by means of the "Set CWS Maintenance Strategy" operation described above.

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 12-bit UH fields of CWS entries. These fields tell us about references to pages during the last 12 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 I 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 every time the process gets dismissed, 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 new 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 performed 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 is 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, DMWS. Setting or resetting DMWS 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 12. An entry which meets this condition is deleted from CWS iff 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 DMWS 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 = Ø. 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 of DMWS. 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.

The page named by PMT entry 105_8 was referenced during

the last interval but 1,

the last interval but 4,

the last interval but 6,

the last interval but 7,

and

the last interval but 8.

A Sample CWS Entry with
USE HISTORY Field Interpreted.

Figure 1