<DOCMPS>DOCSEG.NLS;6, 4-FEB-72 11:46 WHP ;
MPS Segmentation System Overview

All data in the MPS are stored in segments. There are three kinds of segments: file, data, and window.

A file segment is simply the addressable version of a file in the 10X system.

A data segment is the analogy of private data in the 10X virtual memory. Data segments can be named and swapped into and out of virtual memory (VM)

Window segments provide direct, linear addressing of objects composed of pages selected from one or more file or data segments.

At any given moment some set of segments is present in the user's VM ("swapped in").

If the user wants to directly access a segment, it must be swapped in. Also, if a stack (a type of data segment) needs to be lengthened, it may have to be moved in VM. this is equivalent to swapping it out, and then swapping it back in with a different length. Whenever a segment is being considered for swapping or is swapped, a strategy procedure (either system or user supplied) is called to decide whether to swap the segment, or to fix up any addresses which may be necessary (when the segment is actually swapped).

In general, addresses manipulated by MPS programs are in segmented form and require an entire word on the PDP-10 (or two words on a NOVA).

A segmented address contains a segment number, possibly an indirect or index register designator, and a VM address.

Bits	Function
0	0 if in memory, 1 if swapped out
1-12	Segment number
13	Indirect flag
14-17	Index designator
18-35	Address in Virtual Memory (if swapped in), /some invalid, trap-causing address otherwise/

At any given moment, it must be possible to map any VM address into a segmented address and to map any segmented address into a VM address if the segment is swapped in.

When a segment is swapped out, any VM addresses pointing to it must be converted to "swapped-out form." In swapped-out form, any use of a segmented address will cause a segment-out-of-memory signal (signal in the MPL sense) to be generated.

The area of virtual memory in which stack segments may reside is quantized at the 10X page level (512 words). Any private pages in this segmented virtual memory (SVM) are actually pages in a file, called the DATAFILE, created at the beginnning of an MPS session by the segmentation machinery.

A file or a window segment or a data segment of at least one page resides in as many whole VM pages as are needed to hold it while it is swapped in.

Data segments which are less than one page long are grouped together in pages. If one such small segment is swapped in, a request to swap in a small data segment which co-resides in the same page (called a "cosegment") requires only that the segment

be marked as swapped in. If a small data segment is moved (as a side effect of lengthening it, for instance), no attempt is made to reclaim the space previously occupied by it in the current implementation (this may very well change).

An empty page slot in SVM may mean one of two things:

(a) it is an unoccupied slot of a window segment currently covering that page;

(b) it is a genuinely free page of the SVM.

Since the segmentation system is a software administered discipline, swapping of segments in and out of the SVM requires advice from programs outside of the segmenting machinery itself, in general. Such strategy procedures are invoked whenever

(a) a segment is being considered for swapping: if the strategy procedure returns the value FALSE, the segmenting machinery will not swap the segment;

(b) the segment is about to be swapped out; or

(c) the segment has just been swapped into SVM.

The section on swapping below outlines these interfaces in detail.

Segments have names, either user specified or system supplied,and are hierarchically related. The only inviolable rule on segment naming is that no two segments with a common immediate parent may have the same simple name; this prevents ambiguous naming .

(ucalls)

Routines for Manipulating the Segmentation System

Creating and Destroying segments:

SegNum ← NewData(Nwords)

Creates a new data segment of the specified length and returns the segment number.

The segment initially contains only zeroes.

The segment is given an internal name

SegNum ← NewWindow(Npages)

Creates a new window segment with the given number of page slots and returns the segment number assigned to it.

All the page slots are initially empty

The segment name is internal as for NewData

SegNum ← NewFile(Name, Options)

Opens the file with the given name (string), with options as given (this is just currently passed to the 10X system, and is thus highly system dependent), and assigns it a segment number, which is returned. The Options will hopefully be separated from the specifics of the 10X system in a future version of the segmenting machinery.

The segment name is the same as the file name.

DestroySeg(SegNum)

Destroys the given segment and releases the segment number.

If the segment is a data segment, it is obliterated, and the segment number is released.

If the segment is a file segment, the file is deleted (in the 10X sense) and the segment number is released.

If the segment is a window segment, its private pages are destroyed, its directory data segment is destroyed, and the segment numbers of both the window and the directory segment are released.

ReleaseSeg(SegNum)

Behaves like DestroySeg except when SegNum represents a file segment, in which case, the file is deleted from the segmented

address space but is not deleted as a 10X file.

Changing Attributes of Segments

DelPage(SegNum, Page)

Destroys the given page in the given segment.

Cannot be used to delete pages from data segments. These must be contiguous memory -- use SetLength to delete tail of data segment.

If the segment is a file segment, the page is actually destroyed. (However, if the segment is later mapped in, a zeroed page is created in this position).

If the segment is a window segment and the page belongs to another segment, the page is removed but not destroyed. If the page is private to the window segment, it is destroyed.

CreatePage(SegNum, Page)

Creates the given page in the given segment, which must be a window segment. The page must be within the current length of the window. (Use SetLength to adjust the number of pages in the window).

MapPage(SegNum1, Page1, SegNum2, Page2)

Puts page Page1 from segment SegNum1 into the slot at Page2 in segment SegNum2, which must be a window segment.

The same actual page will appear in the two places, not a copy. (this has implications for the NOVA MPL implementation, of course)

If the destination page is occupied, then the prior contents are deleted (via DelPage).

If the source is a data segment, then it must be at least a page long.

SetLength(SegNum, Nentries)

Sets the length of the designated segment as given. It may be necessary to "move" the segment if its length is being increased, in which case it will first be swapped out (by SetLength) and then later swapped in.

Nentries is the number of pages for a window segment or the number of words for a file or data segment.

Lengthening a data segment appends zeroes to it.

Lengthening a file or window segment appends empty pages(slots).

IncLength(SegNum, Nentries)

Increments the length of the designated segment as given.

The effect is exactly:

SetLength(SegNum, ReadLength(SegNum)+Nentries).

Swapping segments and User Control of Swapping

Addr ← SwapIn(SegNum)

The specified segment is swapped into VM and its base address in segmented form is returned as value.

If the segment cannot be swapped in, a signal is generated and passed to the caller.

A strategy procedure is invoked (as described below under SegStrategy, SpSegStrategy, and Lock)

(a) to determine whether it is ok to swap the segment into VM; and

(b) to perform any necessary housekeeping for the segment after it has been swapped in.

SwapOut(SegNum)

The given segment is removed from VM. If the segment cannot

be swapped out, a signal is generated.
A strategy procedure (as described below under SegStrategy, SpSegStrategy, and Lock) is invoked to decide whether the segment is allowed to be swapped out, and if so, the strategy procedure is invoked again just before the segment is actually removed from VM.

Lock(SegNum)
The given segment is locked into place. If it is currently swapped out of VM, it will not be able to be swapped in until an Unlock is performed for it. If it is currently in VM, it will remain at the same VM address until Unlocked.
Locking overrides any other user-defined or system-supplied strategy procedures for the given segment.
Any attempt to lengthen a locked segment will cause a signal to be generated by the system.

Unlock(SegNum)
Allows the given segment to be moved or swapped or lengthened. It does not in any way affect the swapped state of the segment.

SegStrategy(SegType, ProcAddr)
The procedure p pointed to by the ProcAddr parameter will be invoked if a request is made to swap any segment S of type SegType and the following conditions hold:
(a) S is not locked;
(b) either no special strategy procedure has been specified for the particular segment S being swapped, or the value FALSE was returned by the current special strategy procedure for S when called with the ReallyGoing parameter = TRUE (see description of parameters to strategy procedures following).
Whenever invoked, p will be called as
p(SegNum, GoingOut, ReallyGoing), where:
SegNum identifies the segment being swapped,
GoingOut=TRUE means the segment is to be swapped out (GoingOut=FALSE means it is being swapped in),
ReallyGoing=FALSE means the segment is being considered for swapping; if p returns TRUE, the segmentation system assumes that the segment may actually be swapped; if p returns FALSE, the segment will not be swapped.
If ReallyGoing=TRUE, then p should perform whatever pre-swapout or post-swapin housekeeping is required, the direction being indicated by the GoingOut parameter.
If ProcDescriptor=0, the strategy for segments of type SegType reverts to the system default strategy. Presently, this strategy procedure does no housekeeping and simply returns the value TRUE.

SpSegStrategy(SegNum, ProcAddr)
The procedure p pointed to by the ProcAddr parameter will be invoked if a request is made to swap some segment S and S is not Locked, as described above. The arguments to p are as described under SegStrategy.
If some segment S with special strategy procedure Ps is being swapped, and Ps is called with the ReallyGoing parameter = TRUE, Ps may choose to let the strategy for segments of the same type as S be called also. It does this by returning the value FALSE. If the ReallyGoing parameter is FALSE, the value

returned by Ps is used to decide whether or not the segment may be swapped, and no call on the current strategy procedure for
segments of S's type will be made.

Reading Attributes of Segments

SegNum2 ← ReadWindow(SegNum1, Page1: Page2)
Returns the segment and page address of the page (of a data or file segment) residing in the window segment slot, or 0 if the slot is empty.
A returned value of -1 means the page is a data page private to the window segment.

Nentries ← ReadLength(SegNum)
Returns the length of the given segment.
The length of a data or file segment is measured in machine words, and of a window segment in pages.

Type ← ReadType(SegNum : BtMask, XtMask)
Returns the type of the given segment as follows:

Type	Meaning
freebt	Segment does not exist
datbt	Ordinary data segment:
" +stkxt	Stack segment
" +procxt	Process static data segment
" +wsdxt	Window segment directory
filebt	Ordinary file segment
" +codext	Code segment
windbt	Window segment
systyp	For special use by system
" +sptxt	Special strategy procedure

The two parts of the type code may be selected by the two masks BtMask and XtMask which are the second and third values returned, respectively.

SegNum2 ← NextSeg(SegNum1)
Returns the number of the next higher segment above the given one which actually exists, or zero if none.

Page2 ← NextPage(SegNum, Page1)
Returns the number of the next higher page above the given one which actually exists in the given segment, or -1 if no higher page.

Address conversion and analysis

SegAddr ← VToSAddr(VMAddr)
Given a virtual memory address, returns corresponding segmented address or 0 if location is not part of any (swapped in) segment,

Addr ← MakeAddr(SegNum, WordNum)
Creates a segmented address from a segment and word number.

SegNum ← SegNumPart(Addr)
Extracts the segment number from a segmented address.

WordNum ← WordPart(Addr)
Extracts the word displacement from a segmented address.

Flag ← SwappedIn(SegNum)
Returns 1 if the segment is in VM, or 0 if not.

Naming Segments

The segmentation machinery supports an hierarchical naming scheme for segments: a name conflict exists if and only if two segments with a common immediate parent have the same simple name.

SetSegName enforces this rule.

FindSegName(String, ContextSeg)
Returns the segment number of the segment with the given string as name, within the context of the specified segment, or 0 if no match is found. The following rules are used to determine if there is such a segment:
(a) if one of the segments owned by the given segment has a name which matches the given string, that segment's number is returned;
(b) if no segment is found which matches under rule (a), then a search is made starting at ContextSeg and proceeding from parent to parent until a name match is found or the top of the ownership tree is encountered.

SetSegName(SegNum, String)
Sets the name of the given segment to the string. If any sibling of SegNum already has the name String, a signal is generated.

nchar ← ReadSegName(SegNum, String, FirstChar, LastChar)
Reads the name of the given segment into the string starting at position FirstChar. If more than (LastChar-FirstChar+1) characters are needed to contain the name, a signal is generated; as many characters as will fit will already have been placed in the string by the time the signal is generated. A segment which has not been assigned a name by a previous call on SetSegName (or NewFile, which calls SetSegName) has a system supplied name of the form '# .NUM where .NUM is the segment's number.
The number of characters in the name is returned.

Segment Parentage and Associations

MakeParent(SegNum, ParentSegNum)
Make ParentSegNum the parent of SegNum. SegNum will be the "first" child of ParentSegNum. Checks for possible naming conflicts among siblings.

ParentSeg(SegNum)
Returns the segment number of SegNum's parent segment, or 0 if there is none.

ChildSeg(SegNum)
Returns the segment number of the first child of segment SegNum, or 0 if the segment has no children.

SiblingSeg(SegNum)
Returns the segment number of SegNum's sibling, or 0 if he has none. The children of a given segment are "ordered" so that one can sequence through them by acquiring the segment number of the parent's first child using ChildSeg and then moving from one child to the next using SiblingSeg until the family is exhausted.

LinkSeg(fromSegNum, toSegNum)
A "link" to toSegNum will be placed in the segment table entry for fromSegNum (which must not represent a file segment for implementation reasons). This has the following uses:
(a) if fromSegNum represents a stack segment, and toSegNum a process data segment, then the stack segment will be linked to the data segment of the process (in the MPL sense) to which it belongs;
(b) if toSegNum is a code segment, and fromSegNum is a process data segment, then this associates the code segment

with the process.

RdLinkSeg(SegNum)

Returns the segment number to which SegNum contains a "link" (set by AssocSeg).

Implementation

Tables

Core allocation table

Entry per virtual memory page giving:

caseg -- segment number if occupied, 0 otherwise.

casa -- address of segment on data file.

When try to swap in a segment, use this table to find an adequate number of adjacent free pages.

Also used to map virtual memory addresses to segmented memory addresses.

Segment table

One entry per segment, containing:

sttype -- type of segment, (free, data, file, window)

stloaded -- true if segment swapped in to the virtual memory

staddr -- if stloaded then VM address of segment, else datafile address.

stsize -- size of segment in pages for window segment, in words for all other types

stfile -- handle on segment in file system if file segment, segment number of associated segment if stack or data seg, of directory if window segment

plus several other fields, most of which are used to show relationships between segments and are used by the symbol table machinery.

Symbol table

Data file

Management of storage on the file

Large segments -- size >= one page