Section III -- VMS Memory Management

Introduction

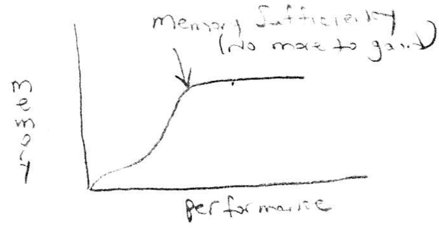
The VMS memory management system is excellently designed, eliminating or diminishing many of the problems found in other architectures (IBM, UNIX, etc.) Some of its characteristics are:

-As on any virtual system, applications should exhibit good program and data locality if satisfactory performance in reasonable amounts of real memory is desired. With VMS, although software with poor locality will itself perform poorly, it will degrade other users less than on other systems.

-The operation of the system is automatic. Applications which must, or even should, directly manipulate any aspect of the memory management system -- page locking, working set size, swapping control, etc. -- are very rare, except in the case of real time procedures.

-The system is tolerant. Once set up ("tuned") only dramatic changes in workload composition require tuning adjustments.

-As the amount of physical memory is increased, performance improves significantly, until a certain amount of memory is installed. At that point additional memory is of no value (and may be harmful). On a 780/8200, this amount is typically around 4 to 8 megabytes, except where poorly designed software is common, or database applications use memory as a disk data cache.



Unfortunately, AUTOGEN sets up VMS in a fashion which thoroughly misuses the power of the memory management system. The effect of this is not necessarily poor performance -- it simply raises the amount of memory needed to achieve the same performance achieved with proper tuning and less memory. Typically, the parameter settings installed by AUTOGEN result in 50% to 200% more memory being required than would be the case for a proper set up of the memory management parameters.

Increases the level of memory sufficiency.

Working Sets

Each VMS process has, at any given point in time, a fixed amount of memory space which may be directly accessed. This space is called the "working set", and is allocated and manipulated in units of "pages" (512 bytes).

The working set has an upper size limit at any given time called WSLIMIT. The actual size of the working set may be below this at times. This value is dynamic and may change several times a second for many processes.

WSLIMIT

Generally, only a small fraction of the memory space which a process may legally access is in the working set at any time.

Shared Memory

Certain pages of memory may be accessible to more than one process at a time. These are called "shared" or "global" pages. These may be from shared data areas, or shared code, either from the shared run-time libraries or sharable programs.

Pages which may be accessed by only one process are called "private" pages.

Shared pages within a working set count the same as private pages towards the WSLIMIT, even if they are actually in more than one working set at the time.

← * Can give you more memory than is actually available.

Page Faulting

If a program attempts to reference a page which is legally accessible but is not currently in the working set, a "page fault" occurs and steps are taken to insert the page into the working set. (The term "page fault" will be used to refer to the entire process of finding and inserting this page.)

-If actual size of the working set is at WSLIMIT, a page is removed from the working set. The page

WSLIMIT

chosen is that page which is not currently valid in the translation buffer and was put in the working set longest ago. It is rarely the page "least recently used".

*basically a FIFO system. - however does check translation cache to see if it was ~~[illegible]~~ recently used. If it was, it looks at the next page up. etc, etc

-The needed page is located. If not currently in physical memory, it is placed there.

-The proper page of physical memory is made directly accessible to the process.

Implications

A page fault consumes system resources. If avoided, faster execution is achieved. The larger a working set, the fewer faults since more pages are directly accessible. Few programs, during any short time interval, access all or even a large fraction of the total pages legally accessible to the program. Therefore, working sets that are a small fraction of the total size of a program can result in relatively small rates of faulting.

Programs which tend to violate the above assumption are often poorly designed and -- in the absence of appropriate improvements -- are ill-suited for virtual systems.

EG BASIC

Program Loading (image) Image Activator

When a program is started, the process has an empty working set (except for portions of the process header and other process permanent structures in P1 space). When the transfer to the program's starting address is made, a page fault occurs. The system then reads the referenced section of the program from the EXE file. If it is a sharable section, and already in memory as a result of an access by another process, the page is inserted into the working set. As additional sections of the program are first referenced, additional faults occur and more of the program is loaded.

— page fault.

As data areas are first referenced, if their initial content is to be non-zero, they are read from the EXE file. If they are to be initially set to zero, or have no defined initial values, a page of physical memory is zeroed and inserted into the working set.

Implications

Even if there were infinite memory and no limit on working set size, substantial numbers of page faults would occur under VMS since page faulting is the program loading mechanism. When utilizing the rate at which faults are occurring to assess memory adequacy and tuning decisions, page faults due to program loading must be disregarded.

Image activation under VMS is an expensive process, and should be minimized. Applications under VMS should not be organized as large numbers of discrete programs which are called successively.

Automatic Working Set Adjustment

It is very difficult to determine the precise size of working set best suited for any program.

- Varies depending on the use of the program
- Varies as the program moves from phase to phase in its functioning.

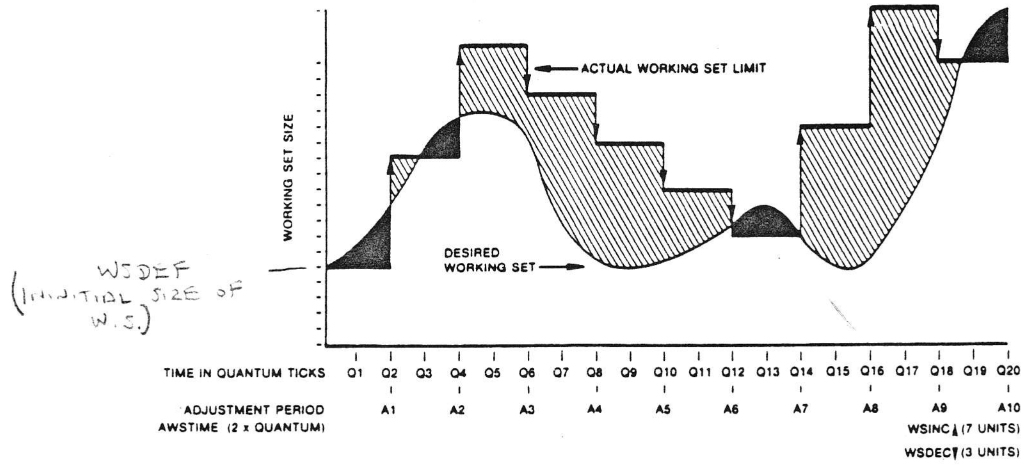
VMS monitors the faulting behavior of each process and adjusts WSLIMIT based on that behavior. This is called "Automatic Working Set Adjustment" or AWSA.

USLIMIT

The algorithm used is far from perfect:

- based on faults per CPU time, not clock time
- doesn't react to severity of current memory load (ie., become more lenient as the load lightens) except for very large working sets
- may produce unnecessary "oscillation" in working set sizes
- is reactive

Figure 3.1
Illustration of AWSA Activity



Despite the above, except in very rare cases, explicitly controlled selection of working set size does not produce better results. AWSA should never be turned off.

<- * check

The procedure used by AWSA

- examines each process at an interval set by AWSTIME as part of QUANTUM end processing
- adds WSINC pages to WSLIMIT if the size of the working set is > 75% of WSLIMIT AND the process has been faulting at a rate greater than PFRATH.

  Note that no pages are actually added to the working set when AWSA increases WSLIMIT. This will happen as further page faulting occurs.
- removes WSDEC pages from the working set and reduces WSLIMIT accordingly if the process fault rate has been < PFRATL.
- will not extend WSLIMIT greater than WSQUOTA (or WSEXTENT if memory demand is light).

~~[illegible]~~ upper L

Implications

Inactive or almost inactive processes are ignored by AWSA, no matter how much memory they are using.

All processes will tend to experience uniform faulting rates except those which expand to WSQUOTA.

Size oscillation can occur, but is seldom a serious problem.

Increasing WSLIMIT is a simple process that consumes no resources.

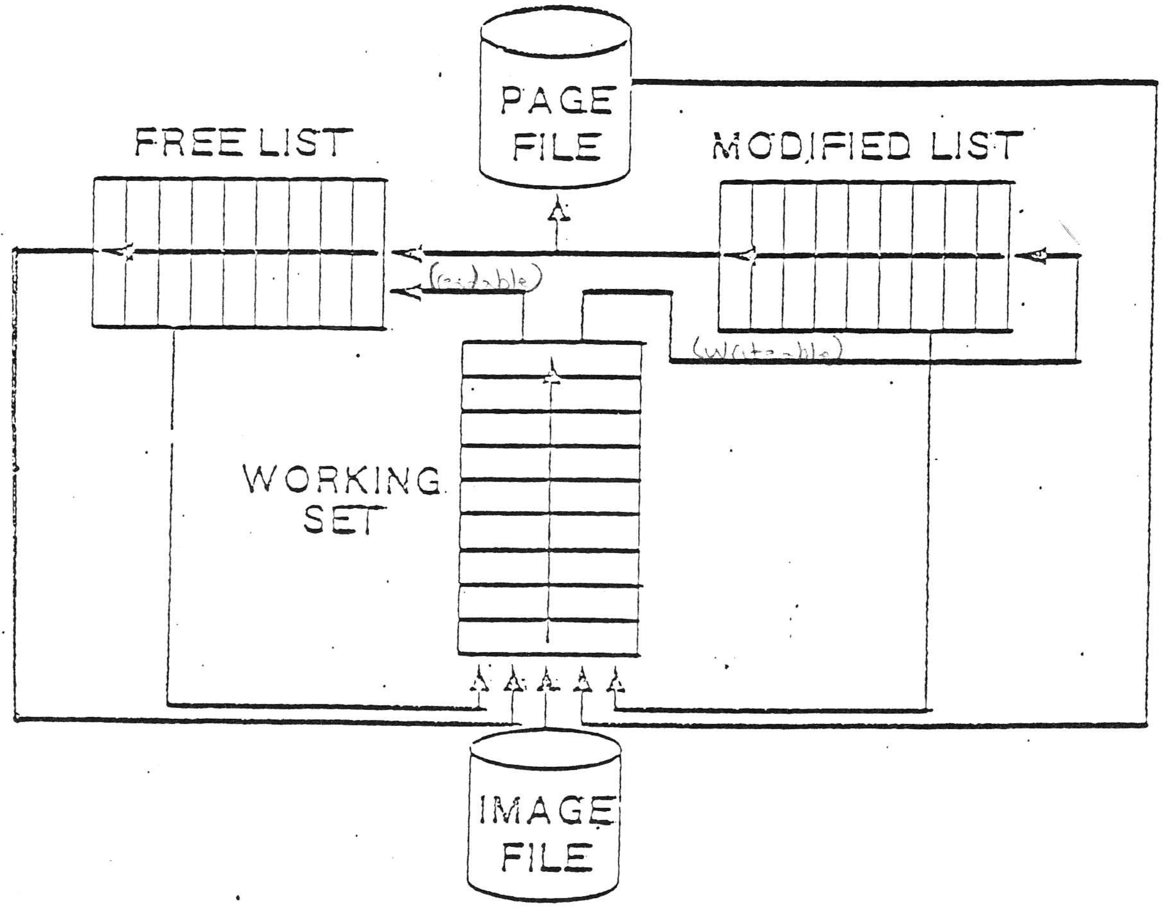
Decreasing a working set size uses some resources, but should not be turned off. If the system has excessive memory, the overhead caused by AWSA can be reduced by decreasing the evaluation frequency (ie., increasing AWSTIME).

Working set sizes are controlled by varying PFRATH and PFRATL, not WSQUOTA. WSQUOTA should be strictly a worst case outer limit.

Because image loading incurs lots of faulting, AWSA will quickly expand a working set to accomodate large programs. WSDEFAULT -- the initial value for WSLIMIT on image activation -- should not be set larger than 100-300, or large amounts of memory and resources will be wasted.

✱ WARNING: The command SET WORKING_SET/NOADJUST turns off AWSA for the process. It is not a privileged command. Users who employ it, and then set a large WSLIMIT for themselves, will wreak havoc on system performance. Best solution -- edit DCL command tables to disable this command.

Figure 3.2
<u>Page Movements Under VMS</u>



The Page Fault Mechanism

Page Storage

Valid pages (ie., those belonging to one or more processes) not currently in any working set are stored in one of 5 places:

EXE disk files

- Read only pages with code or invariant data
- Writable pages with contents initialized to other than zeroes

PAGEFILE.SYS (the system paging file)

- Writable pages

Mapped Section Files (user disk files)

- Pages designated by the process to be written to these files rather than the system paging file when they must be written to disk. May contain initial copies of the page.

Modified Page List (pages in physical memory)

- Writable pages

Free List (pages in physical memory)

- Pages with valid "backing store" -- ie., a current copy of the page exists somewhere on disk (page file, EXE file, or a mapped section file)

The Modified Page List and Free List are known collectively as "secondary cache".

Page Removal From Working Sets

When a page must be removed from a working set because

- a page is about to be faulted in
- AWSA is reducing the working set
- SWAPPER is trimming the working set

if it is

- Sharable, and currently in another working set, the reference to it is simply negated

- Writable, it is added to Modified Page List.

- Read only, it is added to Free List.

None of these steps involve I/O or data movement. They simply require manipulation of pointer tables.

Page Removal From Secondary Cache

The Modified Page List is allowed to get no larger than the parameter MPW_HILIMIT. When it reaches this size SWAPPER is activated and writes pages from it to the paging file or mapped section files until it is down to the size MPW_LOLIMIT.

After writing these pages, they are logically inserted into the Free List.

The Free List has no maximum size. It is always all pages not being used for another purpose. (It has a minimum size; this will be discussed later.)

The name "Free" is a misnomer -- most pages in this list have valid, useful contents. They are only "free" in the sense that, since they have valid backing store, they can be reused for another purpose without worry about saving their current contents.

When physical memory is needed for any purpose, the page at the bottom of Free List (which currently has no valid contents, or is the one placed in the list longest ago) will be selected for the new use.

Insertion of Pages Into a Working Set

When a process page faults, the needed page is located and actions are taken to get the page into the working set. These actions differ depending on where the page is found.

- Page in Secondary Cache (Free List or modified List)

  The pointer table entries are modified to remove the page from the Modified Page List or Free List and include it in the working set.

- Page in another working set (sharable pages)

  The pointer tables are updated to show the page in more than one working set.

- Page never before referenced, zero or undefined initial contents

  The page at the bottom of the Free List is zeroed and added to the working set. This is called a "demand zero fault".

The above three types of page faults are collectively called "soft" faults. The CPU time (780/8200) required to process them is 200 to 300 usecs, except the demand zero fault, which requires 700 usecs. (8600:60 to 80 usecs; 8550/8650: 40 to 50 usecs.) There is no context switch, I/O, or scheduling overhead (except on the 782/8300/8800, where there is if the fault is on the secondary CPU, in which case the processing cost is increased by an order of magnitude or more).

- Page in paging file or a mapped section file (writable pages which have been removed from Secondary Cache)

  The page is read into memory.

- Page in EXE file (page never before referenced, or a read only page which has been removed from Free List)

  The page is read into memory.

The above two types of faults are called **"hard" faults**. Physical memory is acquired by removing pages from the bottom of the Free List. Substantial resources are required:

- 5 milliseconds of CPU time (direct processing cost) (1.5 on the 8600, 1.0 on the 8550/8650)
- CPU slowdown during I/O transfer
- context switch and scheduling processing time
- CPU slowdown due to cache invalidation because of the context switch
- adds to I/O contention
- 20 to 100 milliseconds elapsed time

Page Clustering

When paging I/O is done more than one page is read or written at a time if possible. This grouping is called "clustering".

Writes

Up to MPW_WRTCLUSTER pages can be written in one I/O if contiguous space is in the paging file. Pages are subordered by virtual address contiguity.

*Page files should not get more than 50% F

Reads

Up to PFCDEFAULT pages will be read at one time, but no more than are virtually contiguous and physically contiguous on the disk.

Occasionally a large,. poorly ordered program which tends to run in a small working set will exhibit excessive paging and working set size oscillation due to over-zealous clustering. Relinking the image using a smaller cluster factor or (preferably) explicitly ordering its modules more in line with their usage will result in better performance.

When reads are done from the paging file, clusters of more than a few pages are seldom achieved. If they are, it may be an indication of poor program design.

Implications

The most important goal is to reduce hard faulting because of the large volume of resources it consumes.

Soft faulting is relatively painless -- a soft fault rate of 100/second on a 780/8200 or 500/second on an 8550/8650 adds only 2.5% overhead.

Hard faults are reduced by a large Secondary Cache -- needed pages can then almost always be found via a soft fault. AUTOGEN tuning will allow the Free List to drop to under 100 pages, a value that will guarantee few soft faults and voluminous hard faults unless extremely large working sets are used.

← * problem with Autogen Tuning
Freelist should be 1000 pages or >.

The length of Free List is not the amount of unused memory. In fact, there is seldom any significant amount of memory which VMS allows to be unused, except where the system has a grossly excess amount of physical memory attached. One of the excellent design features of VMS is that it makes effective use of all memory available to it at all times.

The use of mapped section files as an I/O mechanism introduces a large volume of page traffic through the Secondary Caches. This tends to "flush" those caches and, for all system users, dramatically increases hard faulting. Moreover, it is less efficient in directly used resources than simple QIO's. This mechanism has an extremely limited purpose and should be used rarely.

← * WATCH!
- Should stop using RMS
use QIO instead.

EXE files are paging files. They should be as contiguous as possible. Use caution when copying them.

Sharable images, if used by more than one user at a time, reduce hard faults, particularly during image activation. In many instances, this is their only real value; only if there are a large number of simultaneous usrs do they result in significant memory savings.

EDT - INSTALL SHARED

Total page and swap file I/O activity in a properly running system is very low. There is no performance reason to devote a device to these files, severely restrict access to the device containing them, or to install multiple files across different devices. Simply make sure that the device isn't heavily loaded with accesses.

Explicit page locking (by calling the system services SYS$LCKPAG or SYS$LKWSET) seldom improves, and usually harms performance as it makes the page faulting that does occur more costly. It should only be used by real-time, response critical applications.

* don't Lock pages i
the working set. -n
If it is accessed freq
then it will be soft f
back in.

Swapping

The use of memory by VMS will grow as demanded -- AWSA expands working sets as needed; system tables expand as needed; new processes are created on demand (subject to BALSETCNT); and the Modified Page List expands to its upper limit.

Physical memory is finite. If too much is used, it is the SWAPPER's responsibility to free memory.

ANY TIME THE SIZE OF THE FREE LIST DROPS BELOW THE SIZE SET BY THE SYSTEM PARAMETER "FREELIM", THE SWAPPER WILL ACTIVATE AND UNDERTAKE VARIOUS ACTIONS TO FREE MEMORY. Specifically, it will expand the Free List to the size "FREEGOAL".

The steps SWAPPER will take, in priority order, are:

* 1 - Trim the Modified Page List

This does nothing of any real effect on a system with a proper FREELIM size, and should be disabled.

* 2 - Trim to WSQUOTA

Reduce any working set which has been expanded beyond WSQUOTA (see below) back to WSQUOTA.

* 3 - Swap or trim inactive processes

Processes will be selected by the following priorities:

a) "Dormant" processes - any process in COM (computable) state which has not received any CPU time for DORMANTWAIT seconds. Generally, these are batch jobs not running because high priority work is consuming all CPU cycles.

- doesn't work in Vms ver 4 so turn it off.

- swaps out then right ba in.

b) Suspended processes SET PROC/SUSPEND

→ will likely hang system so never use.

c) Processes in LEF or HIB state which have been so more than LONGWAIT seconds.

15 or 20 seconds usually.

4 - Swap or trim active processes

Processes will be selected by the following priorities:

a) Those in CEF state without direct I/O outstanding

b) Those in HIB state or in LEF state without direct I/O outstanding

c) Those in FPG and COLPG states

d) Those in MWAIT state

e) Those in CEF and LEF states with direct I/O outstanding

f) Those in PFW and COM states

Most VMS systems have many inactive processes on them at any given time, and much memory can be freed up by effective swapping. However, swapping or trimming active processes is never desirable. If it happens, it is an absolute indication that re-tuning or additional memory is needed.

Swapping vs. Trimming

Swapping is the action whereby the entire process working set is written, as a whole, to disk. It is accomplished by a few large I/O's directly from the working set to the swap file. A swapped process is not returned to memory until it is re-activated (returns to COM state) even if memory becomes abundantly available. The overhead involved is relatively small, so long as the swap out duration is generally several minutes or more for any given process. Pages freed by swapping are placed at the end of Free List.

Trimming is the action whereby the size of the working set is reduced to an arbitrary size (SWPOUTPGCNT). → def = 60 Trimming inactive processes imposes severe performance degradation on a system.

The pages trimmed go to the top of the Secondary Cache, which tends to push pages from active working sets out of the caches.

Therefore, hard faulting among active processes is significantly increased.

As the trimmed process continues to be inactive, most of its pages cycle out of the Secondary Caches.

The extra volume of pages into the page file tends to increase its fragmentation, possibly decreasing SWAPPER's ability to create large page clusters when writing pages.

When the trimmed process activates again, since 90% or so of its working set has been removed, it will start faulting heavily. Many of these faults will be hard faults. Because clustering from the page file will be poor, many more I/O's will be required to restore the process's working set than if there was an inswap.

Problems With Swapping

A SYS$GETJPI request against a swapped process forces it to be inswapped. A monitoring task which uses GETJPI to make frequent "inspection" passes against all the processes in the system can severely degrade system performance. (MONITOR does not do this.) The exception is a GETJPI call to solely determine the process's state or name.

check this out Watchdog is deadly.

Global pages in a process are not swapped out, they are trimmed. For processes referencing, for example, large global I/O buffers, swapping becomes more like trimming and, therefore, somewhat more costly. However, if the global pages are generally in use by other processes, there is little total system degradation.

Why are we (unthing

① Security (why?) u
should be disciplined.

② Limited no. of Ports (Port contention)

③ - conserve resources? swapped out - so wh bother.

④ Dial-ins - Should Look at how to kick them off if no. activity.

(As of this writing -- VMS 4.4 -- the swapping of Dormant processes by VMS is not functioning due to the fact such processes are scheduled for an immediate inswap after they are swapped. There appears to be an indication that this may not be fixed until version 5.0. Until it is fixed, users should turn off this feature by setting the SYSGEN parameter DORMANTWAIT to a very large value. The number of active batch jobs during times of heavy interactive processing should be held to a minimum. Many sites previously accomplished this by manually suspending batch jobs so they became candidates for swapping -- this must not be used with VMS 4.4 or above because of bugs in the lock manager with shared files.)

Implications

Turn off process trimming! There is no more severe source of poor VMS performance than trimming.

Set FREELIM large. VMS was designed to soft fault efficiently, at the expense of an inefficient hard fault mechanism. If the free list is allowed to get very small, there will be few soft faults and numerous hard faults.

USING MEMORY IS BETTER THAN NOT USING IT. Any use of memory is more productive than having it devoted to inactive processes.

Do not write or use monitors which frequently scan all processes using GETJPI (e.g., SYSDPY, WATCHDOG).

Avoid global buffers if they will not be in active simultaneous use by multiple processes.

WSEXTENT and Page Borrowing

If the size of Free List exceeds BORROWLIM (suggesting very low demand for memory), AWSA expands working sets not just to WSQUOTA but to WSEXTENT. In DEC terminology, this is called "borrowing".

try to set it so that Freelist does not norma
exceed BORROWLIM.

Two safety valves exist to prevent this from over using memory:

The actual expansion of the working set, as opposed to the simple changing of WSLIMIT, to any amount larger than WSQUOTA is inhibited if there are less than GROWLIM pages in Free List.

SWAPPER trims pages used beyond WSQUOTA any time Free List drops below FREELIM.

Implications

A desirable, but hopefully unnecessary feature. Better strategies exist for coping with expected periods of low memory demand. See the section on Tuning Procedures.

Software Design Implications

The VMS virtual memory management has been described as a "Least Recently Used" (LRU) algorithim. By this is meant that it favors access to those memory areas accessed very recently, at the expense of making it very costly to access pages which haven't been accessed in a rather large period of time.

Paqes accessed very recently will typically be found in the workinq set and have a valid entry in the translation buffer -- access is direct with no added cost.

Paqes accessed sliqhtly lonqer ago will be in the workinq set, but not have a valid translation buffer entry. Access requires the added step of fetchinq the address translation entry from the paqe table -- an extra memory reference.

If access has not been for a short while, then the paqe may not be in the workinq set, but is likely to be in the secondary cache, and only the cost of a soft fault is required.

Finally, if the paqe has not been accessed for a considerable period of time, it will not be in memory at all, and the very expensive hard fault processinq must occur.

Software desiqns which access larqe volumes of data must not use virtual memory space inconsistantly with the above facts if good performance is to be achieved. Three typical cases of bad desiqn will be discussed:

Random access over a wide area:

> This is the case where a larqe data area is constructed and accessed in a thorouqhly random pattern -- it is unlikely that any given paqe will, once accessed, be re-accessed relatively soon. VMS will react to this by expanding the working set to fully hold the data area if WSQUOTA permits, utilizinq a large amount of real memory.
>
> A software design which orqanizes the data in a more local fashion, so that once a paqe is accessed, it will probably contain that data which will be needed relatively soon, will function in much less real memory.

Sequential access through virtual memory:

This case involves a large data structure which is accessed sequentially. Portions of it are re-accessed only through another sequential pass. This requires virtually all the re-accesses to be satisfied with a hard fault, as the access pattern is the total opposite of LRU -- the most expensive access pattern possible.

Moreover, as pages of data are used, they are pushed out of the working set and through the secondary cache, tending to displace pages from other users, thus increasing the hard fault rate for all users and significantly impacting total system performance.

The proper design for this type of processing is to use a disk scratch file, accessed via QIO. A relatively small amount of virtual memory is designated as an I/O buffer. As it is re-accessed very frequently, the result is good virtual performance with no impact on other users. That this requires explicit I/O is no drawback -- the QIO will be a considerably cheaper mechanism than hard fault I/O with all its associated table manipulations.

Large structures accessed in portions:

A large data structure is placed in virtual memory and then accessed in small portions -- that is, a small number of pages of data is acted upon exclusively for a period of time, then another small portion and so on.

This tends to result in a volume of hard faulting every time the software switches to work on another portion of the data as it is unlikely that the pages now needed, if they had been referenced before, will still be found in the secondary cache. The pages which had been acted on previously are moved to the secondary cache, displacing pages in use by other users, thus increasing the hard faulting for the system as a whole.

The proper procedure here again is to store the data structure as a whole in a disk file and designate a relatively small amount of virtual memory as an I/O area. Using QIO, the portion of data needed is read from the disk and replaced there (if it has changed) when it is time to move on to another portion. Very few faults will result, there will be no impact on other users, and the QIO's will be very much more efficient than paging I/O. Moreover, since the size of the QIO data transfer can be explictly controlled, either fewer disk accesses will be required, or unnecessary data will not be transferred.