Paging under OS/2

The Life Cycle of a Memory Page in OS/2

The following will attempt to outline the general life cycle of a typical memory page under the OS/2 operating system. A number of constants are accepted as givens for this analysis:

- a memory page under OS/2 is 4 KB in size
- each process under OS/2 has a theoretical, virtual 512 MB address space. Memory allocated by each process is assigned a virtual address within this space by OS/2. This address does not directly correspond to any real, physical address in hardware RAM. When a virtual address is accessed, the x86 CPU translates the virtual address into a hardware address. Since the virtual address space is (often) larger than the amount of physical RAM in the system, the processor may write idle hardware pages out to disk to accommodate new requests. OS/2 supports this aspect of the x86 CPU by means of the "swap file", or the SWAPPER.DAT.
- a 32 bit operating system has a theoretical address space of 4 GB – OS/2 is no exception. The 3.5 GB of virtual memory addresses above the 512 MB for processes is reserved for the system.
- the system, for the purposes of the "life cycle" scenario of the page described below, should be assumed to have plenty of free, physical memory
- by the same token, assume that the page is brand new, never previously used on the system, and is being created in the address space of a newly started process

1. Process starts

The process allocates a block of memory using the DosAllocMem API call.

- OS/2 creates *n*+ "Page Tables", each 4 MB in size, each holding 1024 "Page Table Entries" (PTE). These PTE's correspond to linear addresses, up to 512 MB (or a total of 128 page tables) in the virtual address space of the process. **No physical memory is used at this time.**
- each PTE has a "present" bit, which is a Boolean flag that indicates whether the PTE has been loaded into physical memory. It is clear (FALSE) at creation.
- the number of page tables created by each call to DosAllocMem is dependent on the size of the block of memory that the call is attempting to set aside. This factor is a parameter set in the call to DosAllocMem.
- DosAllocMem returns a pointer to the base PTE

2. Process attempts to use allocated block of memory

The process attempts to read or write to the block of memory at the PTE pointer.

- this causes OS/2 to "commit" the PTE
- OS/2 creates a "Virtual Page" (VP) structure (also 4 KB in size) for each PTE referenced, and updates a pointer in the PTE to point to the VP. This creates a "pointer chain", where the pointer the process has (to the base PTE) leads to a pointer in that PTE to the VP.
- OS/2 does not yet, however, clear the "present" bit in the PTE, **and no physical memory is yet used**.

3. Process reads or writes to block

- this causes OS/2 to try and load the PTE
- if the "present" bit is still clear, this causes a **page fault**
- if the page is present in the file cache, it is taken from there and added to the process's working set. This type of fault causes no disk I/O and is known as a "**soft**" fault
- if the page is not in the file cache, it is read in from disk. This is known as a "**hard**" fault
- in either case, OS/2 now creates a third structure, known as a "Page Frame" (PF) in physical memory and writes the contents of the page into it. This PF is what gets added to the working set of the process that cause the page fault.
- OS/2 updates the pointer in the VP to point to the new PF, which automatically causes a cascading update back down the pointer chain.
- a PF can have one of three states:
    - in-use (it is being used by an active process)

- idle (it is being used by an idle process)
- free (the page has been freed by the process by way of a call to the DosFreeMem API)

4. If memory becomes scarce….

The PF stays in the process's working set until the process deliberately removes it (with a call to DosFreeMem API), or the amount of free physical RAM drops below a pre-defined, system wide threshold. If the latter happens, OS/2 activates a thread called the "Page Ager." The Page Ager proceeds to examine the "accessed" bit and the "dirty" bit of every "in-use" PF in RAM.

- if the "accessed" bit is clear, this means the PF has not been read or written to since the last time the Page Ager ran. If it is not clear, then the page has been read or written to.
- if it is clear, the PF has its status changed from "in-use" to "idle" and is added to the idle list. The process of idling pages in a process's working set is known as "trimming" the working set.
- it stays on the idle list until OS/2 cannot satisfy a page fault request from the free list (that is, the free list is empty – there are no PF's with a "free" status anymore). At this point, OS/2 is forced to take a PF from the idle list to satisfy the new page fault. The oldest PF in the idle list is "reassigned" to the working set of the process that caused the page fault.
- if the "dirty" bit of the PF is set, this means that the PF was written to at some point since it was committed to physical RAM by the original owner process. In this case, the contents of the PF must be swapped out to disk before the PF can be freed and given to the new, page faulting process.
    - if this happens, the PF is written to a "Swap Frame" (SF), which is a 4 KB structure containing the contents of this page. This SF is then written out to the SWAPPER.DAT file, and the pointer in the VP is updated to point to the new SF, instead of the PF. The entire pointer chain is thus updated as well.
    - if the original process requests this page again (i.e., the process regains control of the CPU), the pointer chain starting with the pointer held by the process will lead to this SF, and cause a new page fault, which will cause OS/2 to steal a page from the idle list (perhaps swapping the contents of that PF out to a new SF) and populate it with the contents of this SF. This process can repeat itself indefinitely.

Swap File Sizing

No discussion of paging behavior under OS/2 is complete without a brief examination of how the swap file is dynamically sized to accommodate the demands on the system. The algorithm that is used to grow and shrink the swap file works along the following lines:

- Number of Fixed Pages + Number of Committed Swappable Pages = Total System Pages
- Total Real RAM in System - Total System Pages (size) = Swap File Growth
    - IF Swap File Growth > 0 THEN:
        - SWAPPER.DAT = SWAPPER.DAT + Swap File Growth (rounded to the nearest increment of 512 KB)
    - ELSE IF Swap File Growth < 0 THEN:
        - SWAPPER.DAT = SWAPPER.DAT - Swap File Growth (rounded to the nearest increment of 512 KB)
            - this is accomplished by "marking" an "area" at the end of the SWAPPER.DAT that is a size equal to Swap File Growth. Once all SF's in this area have been freed (this is NOT forced), the SWAPPER.DAT will be shrunk by the amount of Swap File Growth, as calculated above.

Note that this algorithm means that the size **of the SWAPPER.DAT has no direct relationship to the amount of RAM dynamically in use** by any one process. The SWAPPER.DAT is always at least the size of the allocated pages of all processes – this is almost always significantly greater than the actual system wide working set at any given point in time. Thus, the size of the SWAPPER.DAT has no particular relevance for a typical performance evaluation of an OS/2 system.

Flow Diagram of the Process

The diagrams below contain a flow chart of the process described above.