Concept of Virtual Memory - What and Why

In the early days of computers, memories were small & expensive. In those days the programmer spent a lot of time trying to squeeze programs into the tiny memory. Often it was necessary to use an algorithm that ran a great deal slower than another, better algorithm simply because the better algorithm was too big- i.e. a program using the better algorithm could not be squeezed into the computer’s memory.

Basic information

The traditional solution to this problem was the use of secondary memory, such as disk. The programmer divided the program up into a number of pieces, called overlays, each of which could fit in the memory. In 1961 a group of researchers at Manchester, England, proposed a method for performing the overlay process atomically, without the programmer even knowing that it was happening. This method now called virtual memory.

Specification

Virtual Memory (VM) is Not a physical device but an abstract concept Comprised of the virtual address spaces (of all processes) Virtual Address Space (VAS) (of one process) Set of visible virtual addresses
(some systems may use a single vas for all processes)

- **Resident set** - Pieces of a process currently in physical memory
- **Working set** - Set of pieces a process is currently working on

On a machine with VM, the following steps would occur :

1. The contents of main memory would be saved on disk.
2. Words 8192 to 12287 would be located on disk.
3. Words 8192 to 12287 would be loaded into main memory.
4. The address map would be changed to map addresses 8192 to 12287 onto memory locations 0 to 4095.
5. Execution would continue as though nothing unusual had happened.

Principles of virtual memory

System creates illusion of large contiguous memory space(s) for each process Relevant portions of VM are loaded automatically and transparently Address map translates virtual addresses to physical addresses

Single-segment VM:

- One area of 0..n-1 words
- Divided into fix-size pages

Multiple-segment VM:

- Multiple areas of up to 0..n-1 (words)
- Each holds a logical segment (function, data structure)
- Each is contiguous or divided into pages

Main issues in VM design

Address mapping

- How to translate virtual addresses to physical Placement
- Where to place a portion of VM needed by process Replacement
- Which portion of VM to remove when space is needed Load control
- How much of VM to load at any one time Sharing
- How can processes share portions of their VMs

Paging

What is paging?

The idea put forth by the Manchester group was to separate the concepts of address and memory locations. At any instant of time, 4096 words of memory can be directly accessed, but they need not correspond to memory addresses 0 to 4095. For example, we would “tell” the computer that henceforth whenever address 4096 is referred, the memory word at address 0 is to be used. Whenever address 4097 is referred, the memory word at address 1 is to be used. Whenever address 8191 is referred, the memory word at address 4095 is to be used, and so forth. In other words, we have defined a mapping from the address space onto the actual memory address. This technique for automatic overlaying is called paging and the chunks of program read in from disk are called pages.

A more sophisticated way of mapping addresses from the address space onto the actual memory addresses is also possible. For emphasis, we will call the addresses that the program can refer to the virtual address space, and the actual (physical) memory addresses the physical address space.

Implementation of paging:

The virtual address space is broken up into a number of equal-sized pages. Page sizes ranging from 512 to 64 kb per page are common at present, although sizes as large as 4 MB are used occasionally. The physical address space is broken up into pieces in a similar way, each piece being the same size as a page.

Every computer with virtual memory has a device for doing the virtual-to-physical mapping. This device is called the MMU (memory management unit). It may be on the CPU chip, or it may be on a separate chip that works closely with the CPU chip. Let's consider a 32-bit virtual address can be mapped onto a physical main memory address. Since our sample MMU maps from a 32-bit virtual address to a 15-bit physical address, it needs a 32-bit input register and a 15-bit output register.

The 1st thing the MMU does with the page table entry is check to see if the page referenced is currently in main memory. After all, with 2^20 virtual pages and only 8 page frames, not all virtual pages can be in memory at once. MMU makes this check by examining the present/absent bit in the page table entry.

The next step is to take the page frame value from the selected entry and copy it into the upper 3 bits of the 15-bit output register.3 bits are needed because there are 8 page frames in physical memory. The low-order 12 bits of the virtual address are copied into the low-order 12 bits of the output register. This 15-bit address is now sent to the cache or memory for lookup.
As Virtual Memory is too vast topic so it's impossible to describe everything in a single article, hopefully I will post the next article on Virtual Memory soon. If you have any questions regarding VM then don't hesitate simply put it below, I'll try to answer you at my best!

Source: http://www.go4expert.com/articles/concept-virtual-memory-t24269/