4.3. Basic Virtual Memory Concepts

While the technology behind the construction of the various modern-day storage technologies is truly impressive, the average system administrator does not need to be aware of the details. In fact, there is really only one fact that system administrators should always keep in mind:

There is never enough RAM.

While this truism might at first seem humorous, many operating system designers have spent a great deal of time trying to reduce the impact of this very real shortage. They have done so by implementing *virtual memory* — a way of combining RAM with slower storage to give a system the appearance of having more RAM than is actually installed.

4.3.1. Virtual Memory in Simple Terms

Let us start with a hypothetical application. The machine code making up this application is 10000 bytes in size. It also requires another 5000 bytes for data storage and I/O buffers. This means that, to run this application, there must be 15000 bytes of RAM available; even one byte less, and the application would not be able to run.

This 15000 byte requirement is known as the application's *address space*. It is the number of unique addresses needed to hold both the application and its data. In the first computers, the amount of available RAM had to be greater than the address space of the largest application to be run; otherwise, the application would fail with an "out of memory" error.

A later approach known as *overlaying* attempted to alleviate the problem by allowing programmers to dictate which parts of their application needed to be memory-resident at any given time. In this way, code only required once for initialization purposes could be written over (overlayed) with code that would be used later. While overlays did ease memory shortages, it was a very complex and error-prone process. Overlays also failed to address the issue of system-wide memory shortages at runtime. In other words, an overlayed program may require less memory to run than a program that is not overlayed, but if the system still does not have sufficient memory for the overlayed program, the end result is the same — an out of memory error.

Virtual memory turns the concept of an application's address space on its head. Rather than concentrating on how *much* memory an application needs to run, a virtual memory operating system continually attempts to find the answer to the question, "how *little* memory does an application need to run?"

While it at first appears that our hypothetical application requires the full 15000 bytes to run, think back to our discussion in Section 4.1 *Storage Access Patterns* — memory access tends to be sequential and localized. Because of this, the amount of memory required to execute the application at any given time is less than 15000 bytes — usually a lot less. Consider the types of memory accesses required to execute a single machine instruction:

- The instruction is read from memory.
- The data required by the instruction is read from memory.
- After the instruction completes, the results of the instruction are written back to memory.

The actual number of bytes necessary for each memory access varies according to the CPU's architecture, the actual instruction, and the data type. However, even if one instruction required 100 bytes of memory for each type of memory access, the 300 bytes required is still much less than the application's entire 15000-byte address space. If a way could be found to keep track of an application's memory requirements as the application runs, it would be possible to keep the application running while using less memory than its address space would

otherwise dictate.

But that leaves one question:

If only part of the application is in memory at any given time, where is the rest of it?

4.3.2. Backing Store — the Central Tenet of Virtual Memory

The short answer to this question is that the rest of the application remains on disk. In other words, disk acts as the *backing store* for RAM; a slower, larger storage medium acting as a "backup" for a much faster, smaller storage medium. This might at first seem to be a very large performance problem in the making — after all, disk drives are so much slower than RAM.

While this is true, it is possible to take advantage of the sequential and localized access behavior of applications and eliminate most of the performance implications of using disk drives as backing store for RAM. This is done by structuring the virtual memory subsystem so that it attempts to ensure that those parts of the application currently needed — or likely to be needed in the near future — are kept in RAM only for as long as they are actually needed.

In many respects this is similar to the relationship between cache and RAM: making a little fast storage and a lot of slow storage act just like a lot of fast storage.

With this in mind, let us explore the process in more detail.