Memory Management

The memory manager implements virtual memory, provides a core set of services such as memory mapped files, copy-on-write memory, large memory support, and underlying support for the cache manager.

# About Memory Management

Each process on 32-bit Microsoft Windows has its own virtual address space that enables addressing up to 4 gigabytes of memory. Each process on 64-bit Windows has a virtual address space of 8 terabytes. All threads of a process can access its virtual address space. However, threads cannot access memory that belongs to another process, which protects a process from being corrupted by another process.

# Virtual Address Space

The virtual address space for a process is the set of virtual memory addresses that it can use. The address space for each process is private and cannot be accessed by other processes unless it is shared.

A virtual address does not represent the actual physical location of an object in memory; instead, the system maintains a *page table* for each process, which is an internal data structure used to translate virtual addresses into their corresponding physical addresses. Each time a thread references an address, the system translates the virtual address to a physical address.

The virtual address space for 32-bit Windows is 4 gigabytes (GB) in size and divided into two partitions: one for use by the process and the other reserved for use by the system. By default, 64-bit Microsoft Windows-based applications have a user-mode address space of several terabytes.

However, applications can specify that the system should allocate all memory for the application below 2 gigabytes. This feature is beneficial for 64-bit applications if the following conditions are true:

* A 2 GB address space is sufficient.
* The code has many pointer truncation warnings.
* Pointers and integers are freely mixed.
* The code has polymorphism using 32-bit data types.

All pointers are still 64-bit pointers, but the system ensures that every memory allocation occurs below the 2 GB limit, so that if the application truncates a pointer, no significant data is lost. Pointers can be truncated to 32-bit values, then extended to 64-bit values by either sign extension or zero extension.

To specify this memory limitation, use the **/LARGEADDRESSAWARE:NO** linker option. Note that **/LARGEADDRESSAWARE:NO** is ignored for an ARM64 binary. However, be aware that problems can occur when using this option. If you build a DLL that uses this option and the DLL is called by an application that does not use this option, the DLL could truncate a 64-bit pointer whose upper 32 bits are significant. This can cause application failure without any warning.