**Memory – Mapped Files**

Dynamic memory in heaps must be physically allocated in a paging file. The OS’s memory management controls page movement between physical memory and the paging file and also maps the process’s virtual address space to the paging file. When the process terminates, the physical space in the file is deallocated.

Windows memory-mapped file functionality can also map virtual memory space directly to normal files. This has several advantages.

• There is no need to perform direct file I/O (reads and writes).

• The data structures created in memory will be saved in the file for later use by the same or other programs.

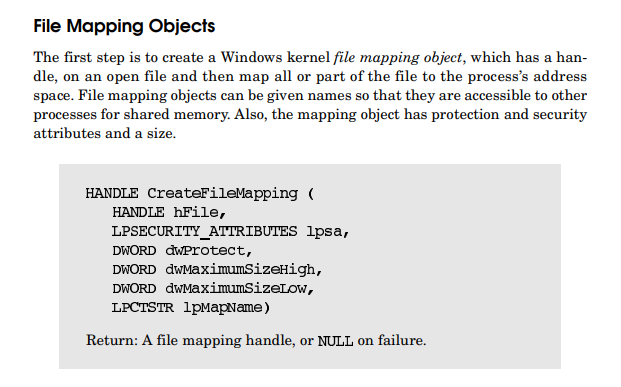
• Convenient and efficient in-memory algorithms (sorts, search trees, string processing, and so on) can process file data even though the file may be much larger than available physical memory. The performance will still be influenced by paging behavior if the file is large.

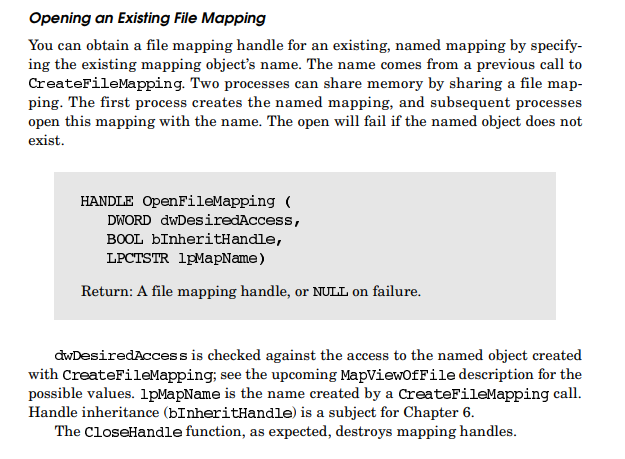
• File processing performance is frequently much faster than using the and file access functions.

• There is no need to manage buffers and the file data they contain. The OS does this hard work and does it efficiently with a high degree of reliability.

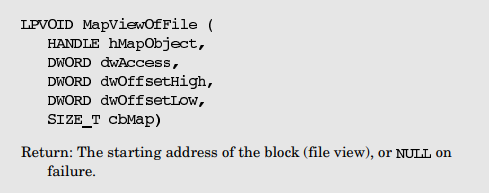
• Multiple processes can share memory by mapping their virtual address spaces to the same file or to the paging file (interprocess memory sharing is the principal reason for mapping to the paging file).

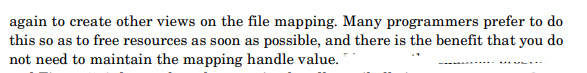
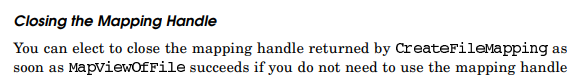
• There is no need to consume paging file space. The OS itself uses memory mapping to implement DLLs and to load and execute executable ( ) files. When reading or writing a mapped file, it’s a good idea to use SEH to catch any exceptions.



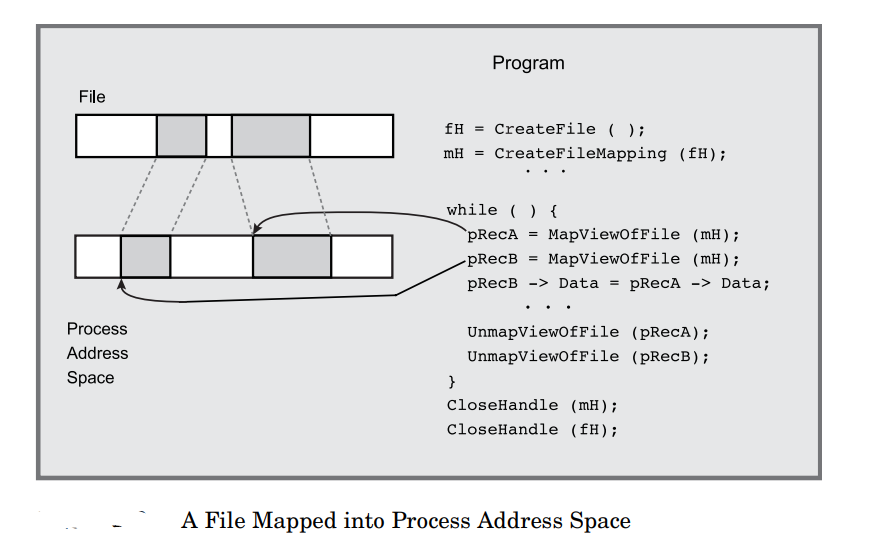


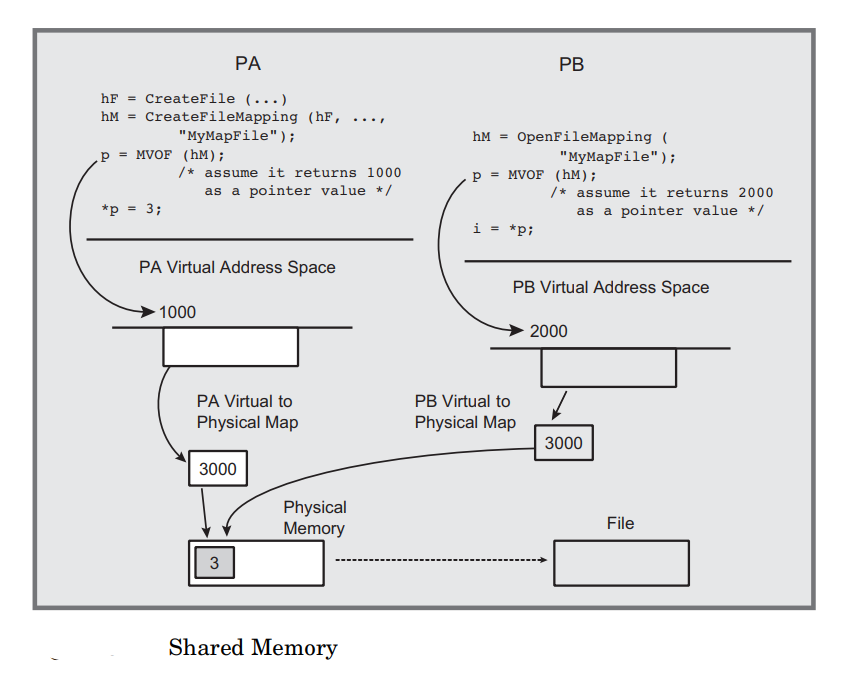


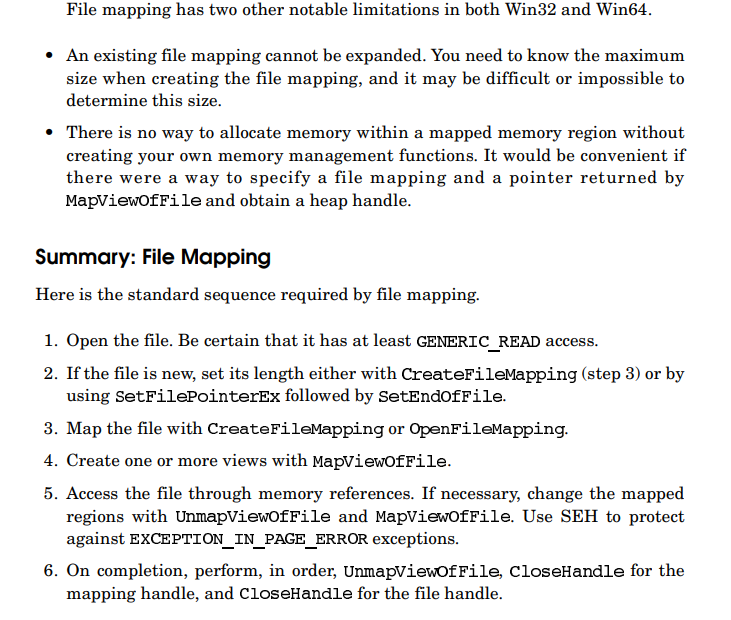












--------------------------------------------------------