Memory Mapping

Memory mapping is a technique that maps files between the logical address space and the physical address space. Also, it provides a mechanism for direct access to files, shared memory regions, or special devices.

The main purpose of memory mapping is:

- Translate from logical address to physical address.
- Memory protection.
- Better management of memory resources.

Advantages of Memory Mapping:

- Faster File Access
- Efficiency
- Efficient Coding Style
- Sharing Memory Between Applications

Benefits of memory mapping:

- Efficiency
- Faster file access
- The ability to share memory between applications
- More efficient coding.

Types of Memory Mapping:

- Direct Mapping.
- Associative Mapping.
- Set-Associative Mapping.

Typical uses of Memory Mapping:

1. Process Loader

Most contemporary operating systems have a process loader, The operating system employs a memory mapped file when a process is launched to load any loadable modules and the executable file into memory for execution.

2. Demand paging

Demand paging is a method used by most memory-mapping systems, in which the file is loaded into physical memory in groups of one page each, but only when that page is referenced. This enables the OS to only load the sections of a process image that need to execute in the specific instance of executable files.

Why we should not use memory mapping instead of buffer pool in DBMSs ?

The mmap has been used in different database implementations as an alternative for implementing a buffer pool, although its design has created a huge number of problems that only can be solved with buffer pool.
A Lot of DBMSs have been depending on the mmap for a long time, which, for other DBMSs, can not work. MongoDB for example, uses mmap for file I/O, but the design had several drawbacks, including:

- complex copying scheme to ensure the correctness
- the inability to perform any compression for data on secondary storage.

Memory Mapping also does not incur the cost of explicit system calls (i.e., read/write), unlike buffer pool. Also, it avoids redundant copying to a buffer in user space because the DBMS can access pages directly from the OS page cache. In addition has many problems that make it undesirable for file I/O in a DBMS. These problems involve both data safety and system performance.

Therefore memory-mapping can be useful for certain file access scenarios, while buffer pool is specifically designed and optimized for DBMS workloads, providing fine-grained control, efficient caching at the page level, support for complex write operations, and optimized memory utilization. Therefore, in a DBMS context, a buffer pool is generally a more suitable choice for caching frequently accessed data pages.