MMAP

[When should I use mmap for file access?](http://stackoverflow.com/questions/258091/when-should-i-use-mmap-for-file-access)

POSIX environments provide at least two ways of accessing files. There's the standard system calls open(), read(), write(), and friends, but there's also the option of using mmap() to map the file into virtual memory.

When is it preferable to use one over the other? What're their individual advantages that merit including two interfaces?

mmap is great if you have **multiple processes accessing data in a read only fashion from the same file**, which is common in the kind of server systems I write. mmap allows all those processes to ***share the same physical memory pages***, saving a lot of memory.

mmap also allows the operating system to optimize paging operations. For example, consider two programs; program A which reads in a 1MB file into a buffer creating with malloc, and program B which mmaps the 1MB file into memory.

If the operating system has to swap part of A's memory out, it must write the contents of the buffer to swap before it can reuse the memory. In B's case any unmodified mmap'd pages can be reused immediately because the OS knows how to restore them from the existing file they were mmap'd from. (The OS can detect which pages are unmodified by initially marking writable mmap'd pages as read only and catching seg faults, similar to Copy on Write strategy).

mmap is also useful for inter process communication. You can mmap a file as read / write in the processes that need to communicate and then use sychronization primitives in the mmap'd region (this is what the MAP\_HASSEMAPHORE flag is for).

One place mmap can be awkward is if you need to work with very large files on a 32 bit machine. This is because mmap has to find a **contiguous block of addresses in your process's address space that is large enough to fit the entire range of the file being mapped**. This can become a problem if your address space becomes fragmented, where you might have 2 GB of address space free, but no individual range of it can fit a 1 GB file mapping. In this case you may have to map the file in smaller chunks than you would like to make it fit.

Another potential awkwardness with mmap as a replacement for read / write is that you have to start your mapping on offsets of the page size. If you just want to get some data at offset X you will need to fixup that offset so it's compatible with mmap.

And finally, read / write are the only way you **can** work with some types of files. mmap can't be used on things like pipes and ttys.