The operating system provides a filesystem interface to load/store program data into a persistent storage device like a disk. Unlike RAM (which is volatile), a disk is a non-volatile storage. By non-volatile, we mean that the data persists even if you power-off your computer. The disk is a very slow device, and frequent reading/writing to disk degrades the performance of the overall system. The operating system provides another interface called memory-mapped files (mmap) that maps the entire file in the RAM. mmap interface returns a virtual address to the programmer that can be used to read/write to the file directly.

Read man pages of mmap and munmap.

So far, we have discussed page tables, virtual and physical addresses. The question is what happens if an application accesses a virtual address which is not present in the page table or does not have adequate permissions. In such scenarios, the x86 hardware generates a page fault. This mechanism allows OS developers to support demand paging. In demand paging, the OS delays the allocation of physical pages until the user application actually tries to access it.

Similarly, the page fault mechanism is required to implement copy on write. In copy on write mechanism, the parent and child processes share the same physical page (with read-only permission). When a parent or child actually tries to write to a virtual page, the OS allocates a new physical page, copy from old physical page to the new physical page, and finally map the pages with the write permissions in their respective page tables.

On x86 hardware, the instructions are complex and can perform multiple operations in a single instruction. After handling the exception, a page fault handler restarts the execution of the excepting instruction. But this is not possible until the hardware provides a mechanism to roll back the changes made by the partial execution of the excepting instruction. The x86 hardware provides precise exception. This property ensures the changes made by a partially executed instruction are rollbacked before the exception handler is called.

Swapping is a mechanism that allows an application to use more memory than the memory available on the system. If a physical page is not available to service a page fault, an already used physical page is saved on a different partition (called swap area), on the disk. When an application tries to access a virtual page that was swapped, the contents of the virtual pages are restored from the swap space.

For the rest of the lecture you can refer to chapter 9 in Silberschatz and Galvin book.