编写程序，连续访问n个数据页，分别具有如下效果：
1、既不触发page fault，也不触发protection exception
2、每次都触发page fault，并从regular file读取相应数据
3、每次都触发protection exception，从anonymous file读取相应数据
评测当n采用不同的值时，上述三种情况产生的时间曲线（无需考虑进程其它代码和准备数据页所需的时间）

会发生 page fault 的情况：

- The page table entry needed for the address translation has zero in its present bit.
- The current procedure does not have sufficient privilege to access the indicated page.

处理器提供给page fault处理函数的信息包括：

- 错误码
- CR2 (control register two). 存储导致page fault的虚拟地址。

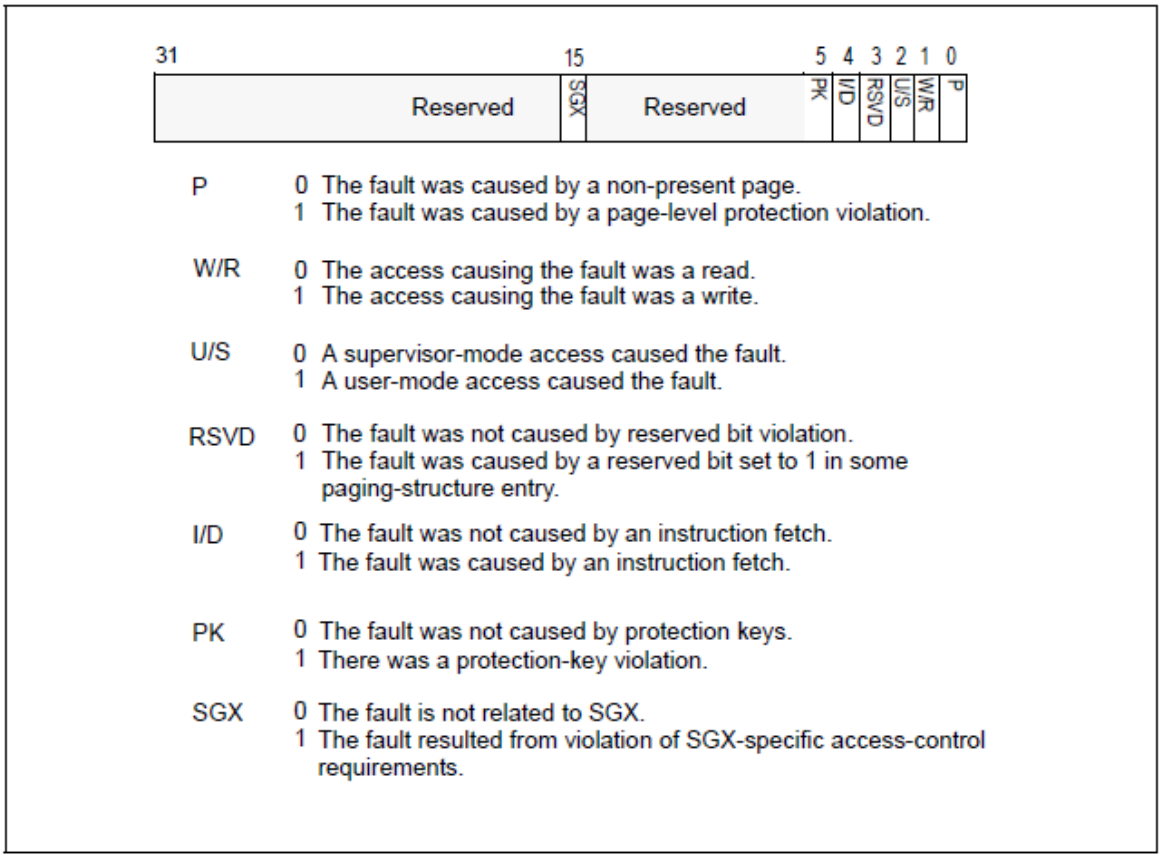

Figure 6-9. Page-Fault Error Code

关于触发protection exception，用COW。和第二问触发page fault区分，管老师给的解释是：原理层面，一个是访问了不存在的页，名称是page fault；另一个是写了只读的页，名称是protection exception。Intel的具体实现，统一是#PF(page fault)入口，然后去判断Page-Fault Error Code。

作业的大致思路：

1. warm up后不会触发page fault
2. mmap建立文件和虚拟地址区域的映射，首次访问该虚拟区域会page fault
3. 访问copy-on-write的页面

实现细节：

1. 第一问：访问mmap的虚拟地址区域两遍，第二遍去计算时间
2. 第二问：第一遍访问mmap的虚拟空间区域时计算时间
3. 第三问可能的思路：mmap的flags参数设为**MAP_PRIVATE|MAP_ANONYMOUS**，先读一遍mmap的虚拟地址区域（热身，保证第二次访问时不发生因为页面不在内存中的page fault），再做写操作（在做写操作时计算时间）关于MAP_PRIVATE参数的解释：Create a private copy-on-write mapping. Updates to the mapping are not visible to other processes mapping the same file, and are not carried through to the underlying file.也就是修改是不会写回到文件中的。
4. 第三问如果用到了fork，在子进程去访问也是可以的。

更多的考虑：

1. 第二问中，如果读连续的页面，可能有“预取”的问题，也就是实际发生的page fault数比预想中的少，如果连续去写呢？
2. 清除已有的缓存干扰。有同学在程序中写了`system("echo 3 > /proc/sys/vm/drop_caches");`，想法是很全面的，不过system的实现是先fork，在fork的子进程中运行execve，实际是清除了子进程的cache。
3. 关于最终结果：第二问和第三问哪种应该更快，助教们没有达成一致，理解作业思路就行。