页式文件系统

- 数据库是能够在大量数据中进行查询和存储操作的系统
- 一个数据表的数据用一个文件来存储
- 文件的存储一般以页为单位(8KB)
- 文件中的页用pageID表示
- [pageID*8KB, (pageID+1)*8KB)

页式文件系统

- 维护若干个存储数据的文件，支持以下操作
  - 创建一个文件
  - 打开一个文件
  - 关闭一个文件
  - 将某个文件中某一页读取到内存中
  - 将某个文件中某一页的内容进行更新

页式文件系统

- 直接利用类似于fopen()、fclose()、read()、write()这些函数进行文件的打开、关闭、读、写操作,读写操作之前需要利用lseek()在文件中进行定位
- 封装成FileMananger类，linux环境可以直接使用,windows环境需要相应的修改

页式文件系统

- 缺点:每次读取和更新文件页的内容都需要进行IO
- 改进方法:利用缓存
- 思路:
  - 需要读取文件页时，先看看该文件页(fileID,pageID)是否在缓存中(hash表)
  - 如果在缓存中，则直接读取相应缓存页中的信息
  - 如果不在缓存中，那么在有限的缓存中申请一个页的空间，利用FileManager类将文件中的页读到申请的缓存页中
  - 需要更新某个文件页中的内容时，也做类似的处理

缓存

- 替换
  - 申请缓存页时，如果缓存满了，需要将一个缓存页“替换”到文件中，腾出空间
  - 采用LRU算法选择替换算法，LRU算法会根据缓存页的访问情况进行选择
  - LRU: Least Recently Used，即替换最近最久未使用页

缓存

- 标记脏页
  - 如果更新了缓存页中的数据，BufPageManager并不会马上将其对应的数据写到文件中，会在将来对该页可能进行的“替换”操作时，将页面写回文件，该页就成为了“脏页”
  - BufPageManager通过一个脏页数组bool dirty[]来记录某个缓存页是不是“脏页”

缓存

- 在FileManager的基础上增加缓存
- 封装成BufPageManager类，linux环境可以直接使用，windows环境需要相应的修改

页式文件系统

- 建议：
- 在利用BufPageManager时，建议FileManager只维护一个打开的文件
- 如果确信对某个缓存页进行若干次访问或更新操作过程中，该缓存不会被替换出去（单线程数据库），那么可以在这些操作结束后在调用access或者markDirty函数，节省时间

页式文件系统

- 说明：
- FileManager类在fileio/FileManager.h中
- BufPageManager类在bufmanager/bufmanager.h中
- 使用时注意include对应的头文件，这两个类的所有实现都在头文件中，所以不需要链接动态库
- 需要引入的文件夹：fileio、bufmanager、utils

TODO

- 如果你想应用这个代码，请完成BufPageManager中getPage和writeBack的代码填空，函数描述已经在代码中给出详细注释
- 这一部分框架的测试代码在testfilesystem.cpp中，假如填空后可以跑通测试代码，输出与refer.out一致，即基本视为该部分功能完整
- Good Luck！