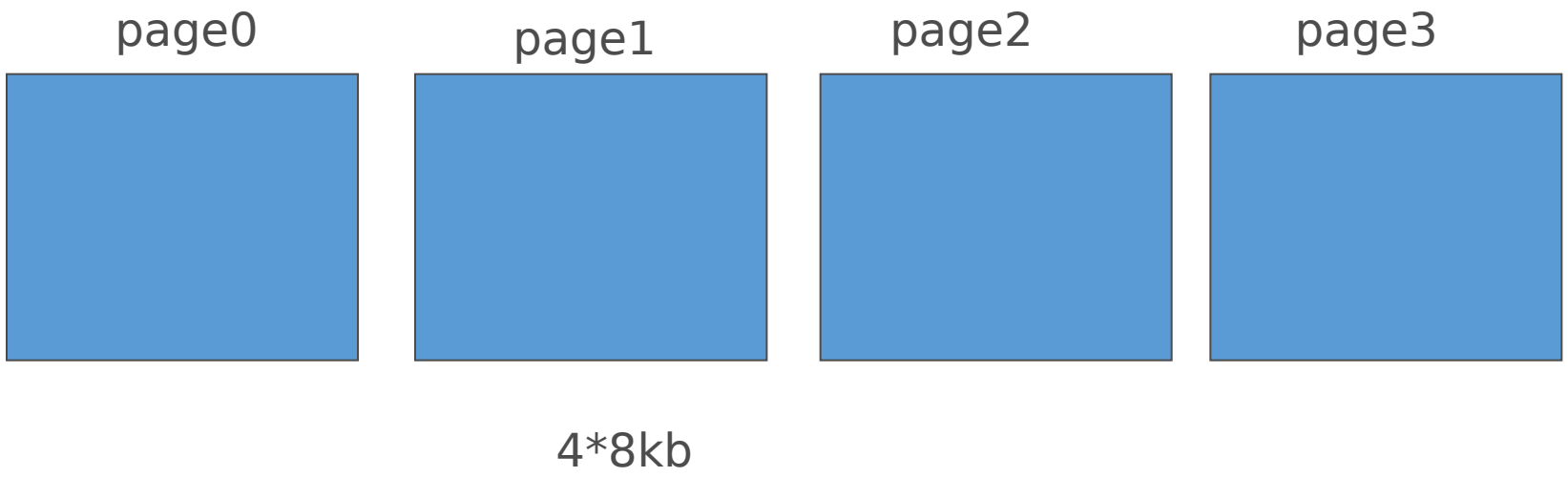
页式文件系统

- 背景1
  - 文件存储一般以页(扇区)为单位(8KB)
  - 文件中的页用pageID表示
  - [pageID*8KB,(pageID+1)*8KB)



页式文件系统

- 背景2
  - 数据库，能够在大量数据中进行查询和存储操作的系统
  - 一个数据库的数据用一个文件来存储
  - 数据库系统可能要维护若干个数据库
  - 问题：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

- 方法1
  - 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

- 方法1、一些概念
- fileID:一个int整数，用来区别程序在运行时通过FileManager类打开的所有文件(不包括关闭的文件)。
- pageID:文件的页号，对应的存储空间为文件中[pageID*8KB, (pageID+1)*8KB]
- (fileID,pageID)就能指定一个文件页面

页式文件系统

- 方法1、利用FileManager类的成员函数
- bool createFile(const char* name)
  - 新建一个文件，文件名由name指定
  - 返回是否新建成功
- bool openFile(const char* name, int& fileID)
  - 打开一个文件，文件名由name指定
  - 返回是否打开成功
  - 如果打开成功，为fileID分配一个文件号
- int closeFile(int fileID)
  - 关闭一个文件，由fileID指定
  - 如果成功，返回0

页式文件系统

- 方法1、利用FileManager类的成员函数
- int writePage(int fileID, int pageID, BufType buf, int off)
  - 更新文件中某一页的内容
  - fileID，pageID指定文件页面
  - buf是一个无符号整数类型的指针
  - buf+off是要写入内容的首地址(无符号整数类型指针)
  - 如果正常写入，返回0
  - 如果pageID超过了文件大小，那么writePage会对文件进行扩充

页式文件系统

- 方法1、利用FileManager类的成员函数
- int readPage(int fileID, int pageID, BufType buf, int off)
  - 读取文件中某一页的内容
  - buf是一个无符号整数类型的指针
  - buf+off表示要将文件页中的内容读到哪里
  - 如果正常读取，返回0

页式文件系统

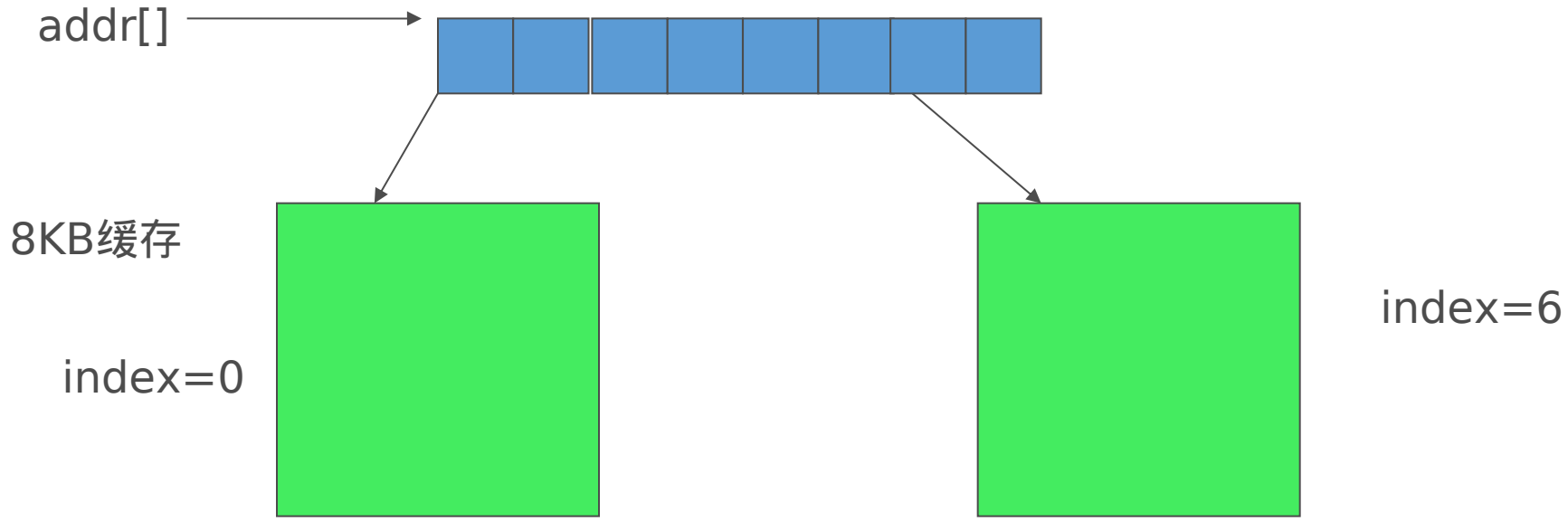
- 缺点:每次读取和更新文件页的内容都需要进行IO
- 改进方法:利用缓存
- 思路:
  - 需要读取文件页时，先看看该文件页(fileID,pageID)是否在缓存中(hash表)
  - 如果在缓存中，则直接读取相应缓存页中的信息
  - 如果不在缓存中，那么在有限的缓存中申请一个页的空间，利用FileManager类将文件中的页读到申请的缓存页中
  - 需要更新某个文件页中的内容时，也做类似的处理
- 注意:缓存页个数有限，申请缓存页时，如果申请的缓存页的个数达到上限，需要选中一个缓存页，将内容写到文件中，腾出该缓存页的空间。这个操作称作“替换”

页式文件系统

- 方法2
  - 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

- 方法2、一些概念
- 缓存页面数组 BufType addr[CAPACITY]
- typedef (unsigned int *) BufType;
- 缓存页面数组下标 index (这个index不是数据库索引)



页式文件系统

- 方法2、一些概念
- 标记访问
  - 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
  - 究竟选择哪个缓存页进行“替换”呢
  - BufPageManager会采用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

页式文件系统

- 方法2、一些概念
- 标记脏页
  - 如果想更新(fileID,pageID)文件页中的数据，BufPageManager并不会马上将其对应缓存页中的数据写到文件中
  - 会在将来对该页可能进行的“替换”操作时，将页面写回文件
  - 但是在替换某个缓存页时，如果该页没被更新过(之前只是读取信息)，那么就没必要写进文件
  - BufPageManager通过一个脏页数组bool dirty[]来记录某个缓存页是不是“脏页”

页式文件系统

- 方法2、利用BufPageManager类的成员函数
- BufPageManager(FileManager* fm)
  - 构造函数
  - fm是FileManager指针，BufPageManager将利用FileManager提供的成员函数进行文件读写操作
- void getKey(int index, int& fileID, int& pageID)
  - 获取缓存页面对应的文件页面
  - index是缓存页面在BufType addr[]中下标
  - fileID和pageID在函数返回时存储文件id和文件页号

页式文件系统

- 方法2、利用BufPageManager类的成员函数
- void writeBack(int index)
  - 将addr[index]指定的缓存页数据写回文件
  - 一般用不着，在程序退出的时候可以调用一下
  - 是不是真的写回则要看dirty[index]是否为true
- void close()
  - 将所有缓存页的内容写回，通过调用writeBack函数实现
- void release(int index)
  - 放弃addr[index]中的内容，将该缓存页置于空闲，没用上

页式文件系统

- 方法2、利用BufPageManager类的成员函数
- BufType getPage(int fileID, int pageID, int& index)
  - 根据(fileID,pageID)文件页找到对应的缓存页面，如果缓存中没有就申请一个缓存页面，并将文件页面中的内容读取到该缓存页面中
  - 函数的返回值就是缓存页面的首地址
  - index是输出参数，记录缓存页面在BufType addr[]中的下标

页式文件系统

- 方法2、利用BufPageManager类的成员函数
- BufType allocPage(int fileID, int pageID, int& index, bool ifRead = false)
  - 同getPage类似，但是不在缓存中进行查找，而是直接为文件页面申请缓存，并根据ifRead是否为true决定是否将文件页中的内容读入缓存中
  - 调用前必须确保(fileID,pageID)没有对应的缓存页，否则会出错
- void access(int index)
  - 标记访问
- void markDirty(int index)
  - 标记脏页

页式文件系统

- 方法2、几个建议
- 在利用BufPageManager时，建议FileManager只维护一个打开的文件
  - 一个数据库的索引和数据都存储在一个文件中，用use命令打开一个数据库时就把上一个关掉
- 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/BufPageManager.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
- 例子