实验二: 缓冲区管理器

主讲教师: 邹兆年(znzou@hit.edu.cn)

姓名：刘璟烁 学号： 日期：2021年5月16日

1. 实验要求

1. **实验目的**

BadgerDB实验的目标是提供关于RDBMS的关键组件的实践训练，在此实验中，需要在实验所提供的存储管理器之上实现一个缓冲区管理器。

1. **实验任务**

本次实验需要补充完整BadgerDB数据库系统项目的整体内容，该项目的最底层是I/O层。该层允许系统的上层创建/销毁文件、分配/取消分配文件中的页面以及读取/写 文件的页面。该层由两个类组成：一个文件（class file）和一个页面（class page）类。这些类使用C异常来处理任何意外事件的发生。异常及上述两个类的实现均以提供。

在此基础上，需要实现其上层：数据库缓冲区(buffer pool)的管理器buffer manager。数据库缓冲区是一个固定大小的内存缓冲区数组，用于保存已从磁盘读入内存的数据库页面（也称为磁盘块）。一个页(page)是磁盘与主存中的缓冲池之间的传输单位。主存中的数据库页面是第一次读取时相应页面的副本。一旦页面从磁盘读到缓冲池，DBMS系统可以更新存储在缓冲区上的信息使用缓冲区中的副本不同于磁盘上的副本，这样的页面被称为“脏页”。缓冲区管理器用于控制内存中的哪些页面。每当缓冲管理器接收到数据页面的请求时，缓冲管理器检查请求的页面是否已在构成缓冲池的frame之一。如果是，缓冲区管理器只需返回一个指向页面的指针。如果没有，缓冲区管理器释放一个frame(如果页面很脏，可能是通过写入页面到磁盘)，然后在请求的页面中从磁盘读取到已释放的frame中。

2. 实验准备

根据实验指导书要求，下载并使用文档生成工具Doxygen为项目生成更易于分析的说明文档，进入bufmgr文档并应用make doc指令：

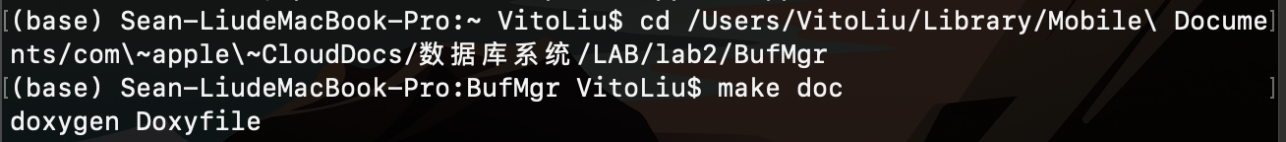


图1. 执行make doc指令

执行成功：

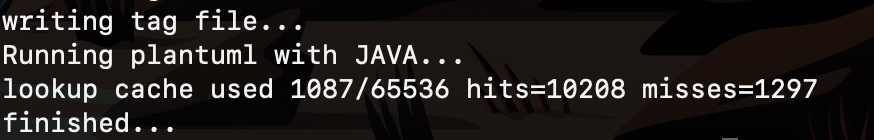


图2. 执行成功

打开在doc目录中出现的index.html文档，可以看到清晰的项目结构：

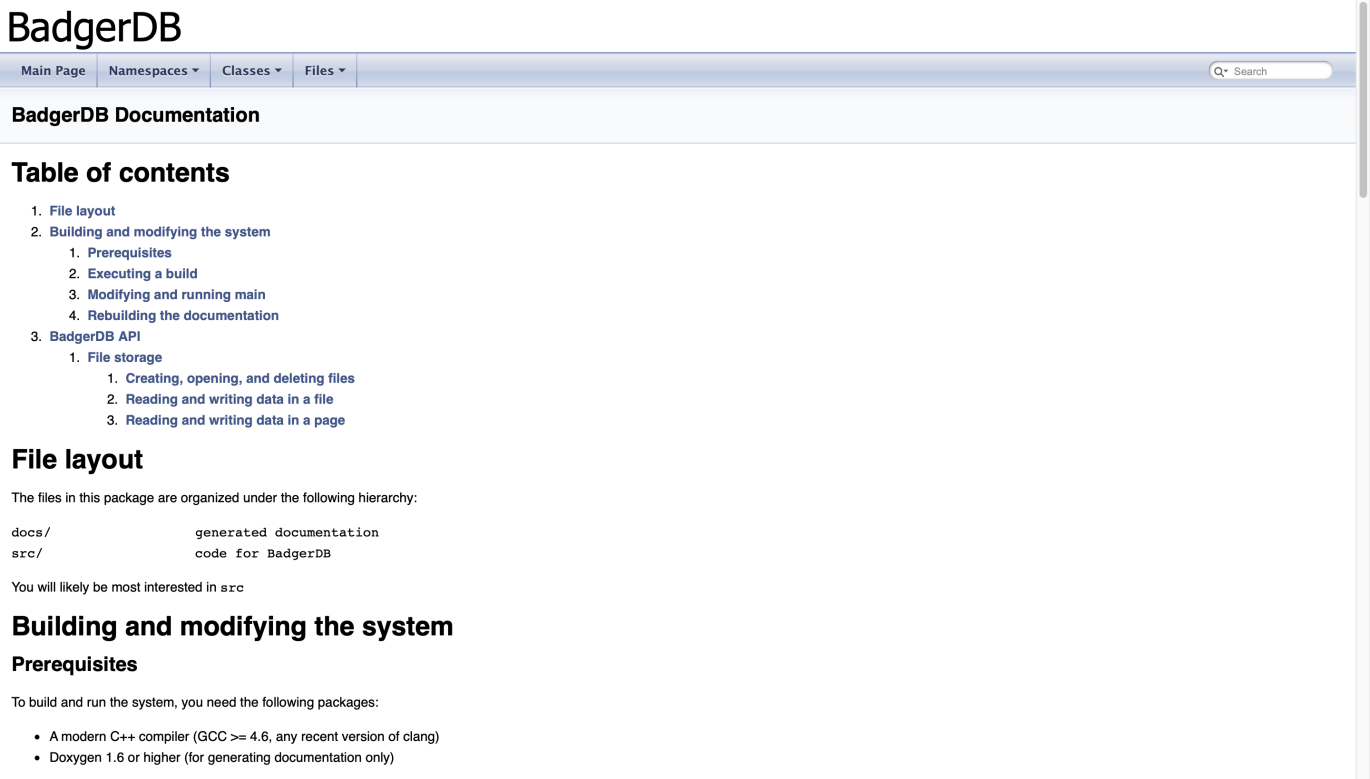


图3. index.html页面

1. 实现过程
2. **实现原理**

Buffer manger 的核心是缓冲区替换策略，本次实验使用时钟算法进行页面的替换，相比LRU它是一种行为接近并且性能更好的策略。

图为对于时钟算法使用缓冲池的概念性布局。其中，每个方框都对应于缓冲池中的一个帧。假设缓冲池包含numBufs个帧，则其编号为0到numBufs-1。缓冲池中的所有帧都排列在一个圆形列表中。每一帧有一个refbit。每次访问缓冲池中的一个页面时(通过调用readPage()方法），该帧refbit置为true。在任何时间点，时钟指针（值介于0和numBufss-1之间的整数）都向前一步（使用取模运算)。对于时钟指针经过的每帧，检查并清除refbit。如果refbit为false，则选择该页面进行替换。如果所选的缓冲区帧为脏页（即已被修改），则当前占用该帧的页面将被写回磁盘。否则，该帧被清除并且一个新的页从磁盘被读取到该位置。

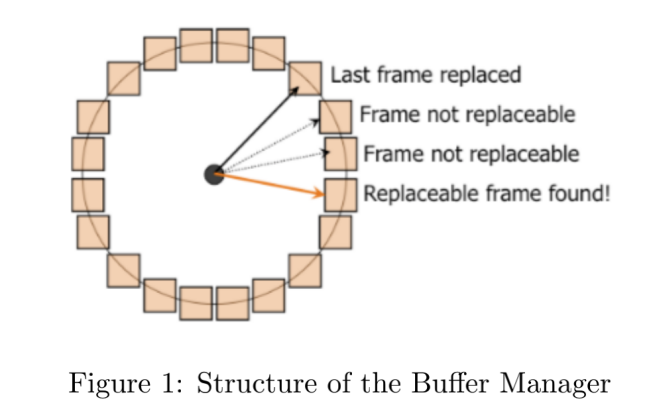


图4. 缓冲区管理器结构

算法的程序框图如下：

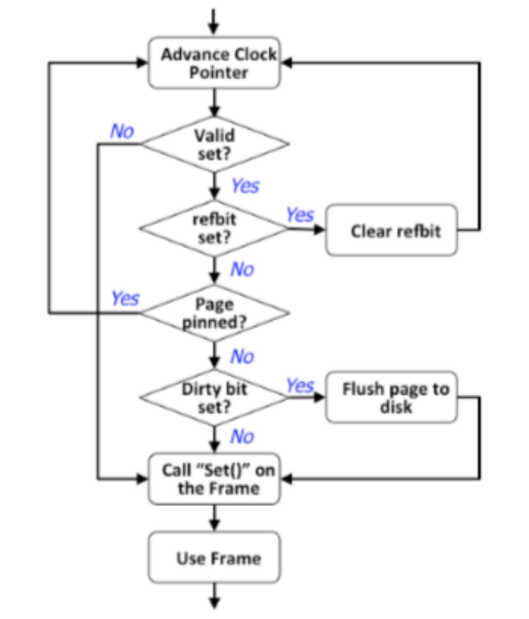
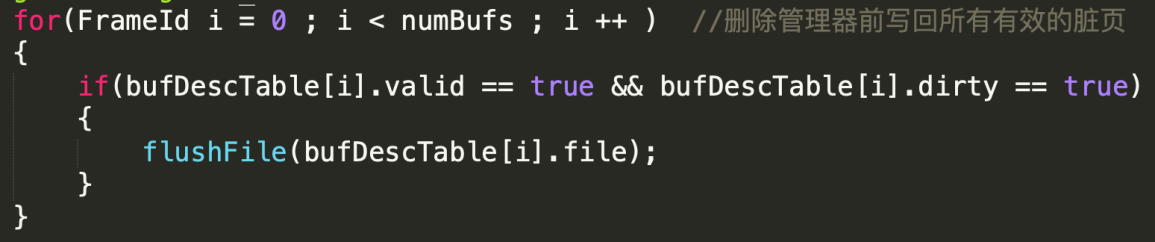


图5. 时钟算法程序框图

1. **函数实现**

* ~BufMgr()

循环便利属性bufDescTable来清除、写回所有有效脏页：



并释放缓冲池、hashTable和BufDescTable。

* void advanceClock()

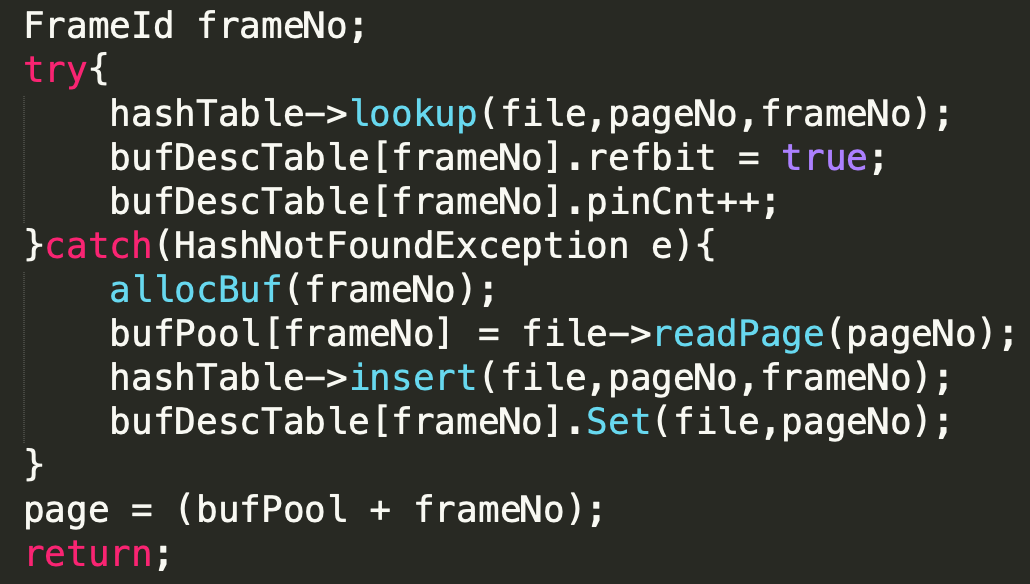
对clockHand加一并对buf数进行模运算即可；

* void allocBuf(FrameId& frame)

按照上面图5所示的程序框图进行实现，并注意如果所有缓冲区帧pinned位均为true，抛出相应异常。将如果分配的缓冲区帧中包含有效的页面，则可以从哈希表中调用remove方法删除相应的条目;

* void readPage(File\* file, const PageId PageNo, Page\*& page)

调用lookup()方法来检查页面是否已经在缓冲池中；当页面不在缓冲池中时，抛出HashNotFound异常并调用上述allocBuf()方法获取一个帧，调用readPage()，将页面从磁盘读入缓冲池帧。随后将页面插入hashTable。最后，调用帧上的Set()将页面的pinCnt置为1。通过页面参数返回一个指向包含页面的帧的指针;否则设置refbit，增加页pinCnt，然后通过页面参数返回指向包含页面内容的帧的指针：



* void unPinPage(File\* file, const PageId PageNo, const bool dirty)

减少包含对应输入页的帧的pinCnt，如果dirty为true，则设置脏位。如果pinCnt计数已为0，则抛出相应异常。如果在hashTable中没有找到页面则不做处理；

* void allocPage(File\* file, PageId& PageNo, Page\*& page)

首先调用allocatePage()方法，在指定的文件中分配一个空页。然后调用allocBuf()分配一个可用帧。随后将一个条目插入hashTable中，并在帧上调用Set()以正确设置它，page将被修改：

截屏2021-05-26 上午11.10.02

* void disposePage(File\* file, const PageId pageNo)

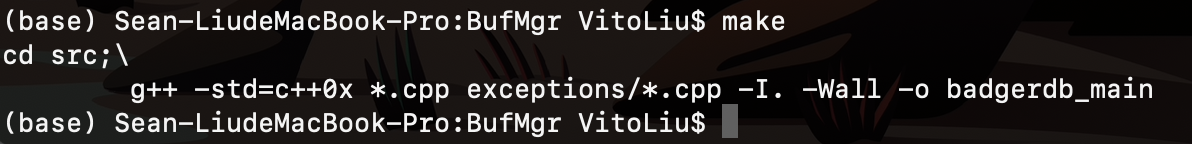
从文件中删除一个特定的页面，在从文件中调用deletePage()删除该页面之前，确保如果在缓冲池中分配了一个帧，则该帧将被释放并删除hashTable中对应项；

* void flushFile(File\* file)

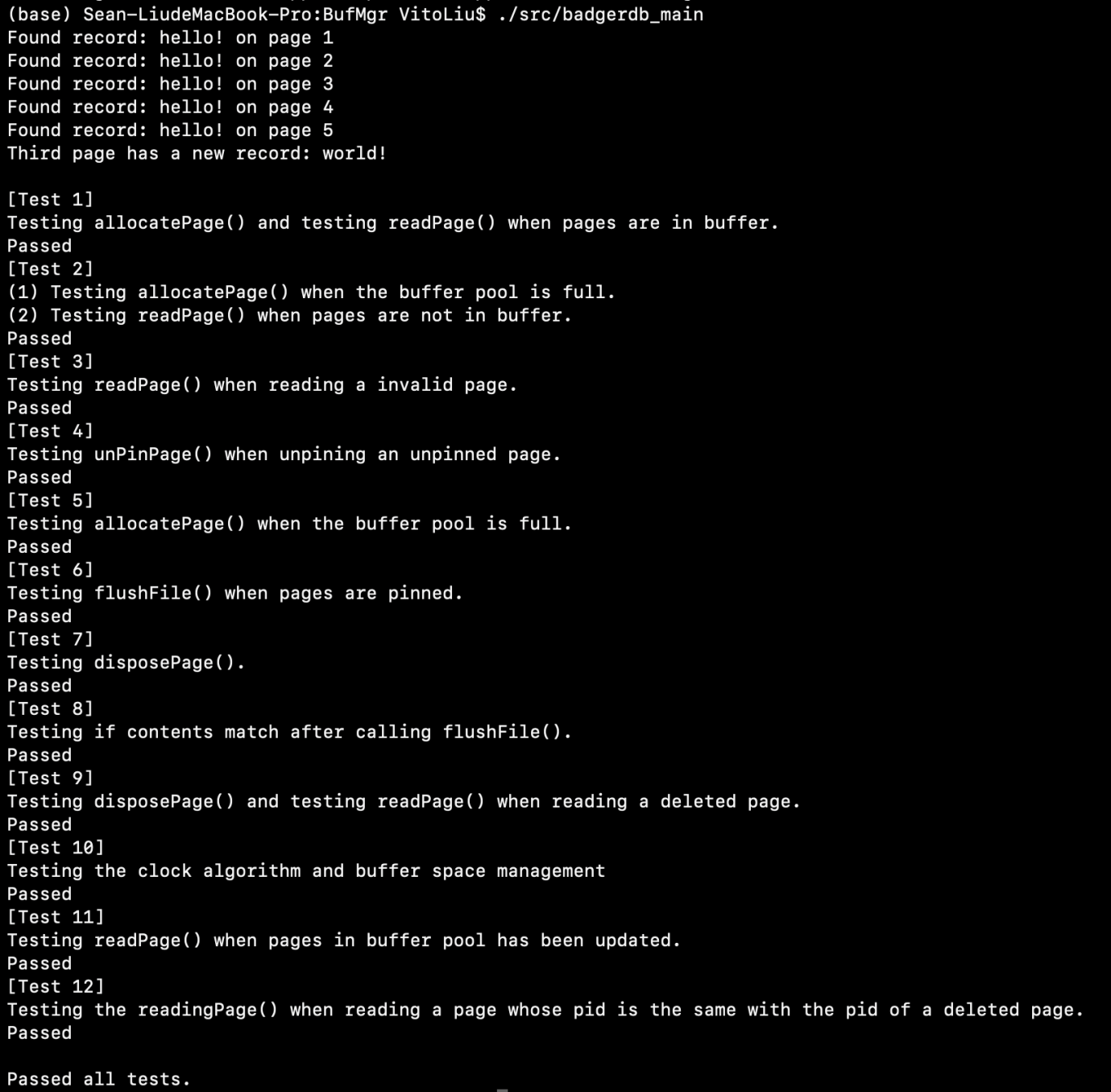
扫描bufDescTable中的属于该file的页，对于遇到的每个页，如果页面很脏，调用file->writePage()将页写到磁盘，然后设置脏位为false；随后hashTable中删除页面（无论页面是干净还是脏页），并为页面框架调用bufDescTable的Clear()方法。如果某些页pinned为true或为无效页，则抛出相应异常。

1. 实验结果

首先输入make指令对项目进行编译：



运行可执行文件,可以看到正确通过了所有测试：



5. 总结与心得

本次实验在提供的存储管理器之上实现一个基于时钟算法的缓冲区管理器，根据实验指导书所给的提示，分别在I/O层基础上实现了向磁盘写回相应页、基于时钟算法在Buffer Pool中替换并分配空间，从而从磁盘读取相应页等方法；进一步掌握并理解了DBMS的缓冲区管理的策略及思想，对于pinned、dirty、refbit等概念在应用的过程中有了更深刻的理解。