实验2: 缓冲区管理器实现(2022春)

主讲教师: 邹兆年(znzou@hit.edu.cn)

1 实验目的

1. 掌握数据库管理系统的存储管理器的工作原理。
2. 掌握数据库管理系统的缓冲区管理器的工作原理。
3. 使用C++面向对象程序设计方法实现缓冲区管理器。

2 实验准备

我们首先介绍本实验提供的BadgerDB数据库管理系统的存储管理器和缓冲区管理器的原理及源代码结构。

2.1 BadgerDB的I/O层

BadgerDB数据库管理系统的最底层是I/O层，它为系统上层提供了创建/删除文件、分配/释放文件页面、读/写文件页面等功能。I/O层由2个C++类实现，分别是文件类(File类)和页面类(Page类)。这两个类使用C++异常来处理系统运行过程中发生的异常事件。本实验提供了File类、Page类及各种异常类的实现。

2.2 BadgerDB的缓冲区管理器

数据库缓冲池(buffer pool)是由一组固定大小的内存缓冲区(buffer)构成的数组，用于存放从磁盘读入内存的数据库页面(page，也称作磁盘块)。缓冲池中每个固定大小的内存缓冲区称作页框(frame)。页面是磁盘和缓冲池之间数据传输的基本单位。大多数现代数据库管理系统使用的页面大小至少为8KB。当磁盘上的页面被首次读入缓冲池时，缓冲池中的页面和磁盘上对应页面一模一样。一旦DBMS修改了缓冲池中该页面的内容，则缓冲池中的页面与它在磁盘上对应的页面就不再相同了。我们将缓冲池中被修改过的页面称为"脏"页面。

磁盘上的数据库通常比缓冲池大，因此任何时候只有一部分数据库页面可以被读入缓冲区。缓冲区管理器(buffer manager)用于控制哪些页面驻留在缓冲池中。每当缓冲区管理器收到了一个页面访问请求，它会首先检查被请求的页面是否已经存在于缓冲池的某个页框中。如果存在，则返回指向该页框的指针；如果不存在，则缓冲区管理器会释放一个页框(如果页框中的页面是脏的，则需要将该页面先写回磁盘)，并将被请求的页面从磁盘读入刚刚释放的页框。

2.2.1 缓冲区页面替换策略

当需要从缓冲池获取一个空闲页框时，有很多方法来确定替换掉缓冲池中哪个页面。常用的策略包括我们课上学习过的FIFO、MRU和LRU。尽管LRU是最常使用的策略之一，但它的开销大，并且在数据库系统运行的很多常见情况下，LRU都不是最优的策略。于是，很多系统采用时钟算法(the clock algorithm)来近似LRU的行为。时钟算法的执行速度非常快。

图1给出了概念上的缓冲池布局，其中每个正方形表示一个页框。假设缓冲池中包含numBufs个页框，编号从0到numBufs − 1。所有页框在概念上被组织成一个环形列表。每个页框带有1位，称作refbit。每当缓冲池中一个页面被访问了(通过调用缓冲区管理器的readPage()函数)，则该页框的refbit被置为1。表针指向缓冲池中的一个页框，实现上用一个0到numBufs − 1之间的整数来记录被表针指向的页框。表针顺时针转动，其实现方法是将表针指向的页框号变量加1，再对numBufs取模。每当表针指向一个页框时，算法检查该页框的refbit的值，并将refbit置为0。如果refbit此前为1，则该页框中的页面"最近"被访问过，因此不替换该页框中的页面；如果refbit此前为0，则选择该页框中的页面进行替换(假设该页面是没有被固定

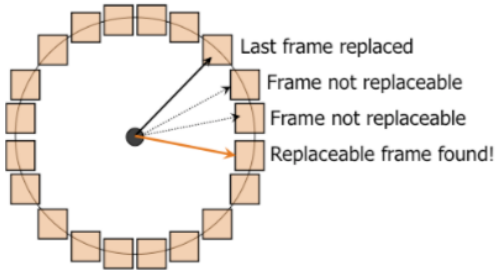

Figure 1: Structure of the Buffer Manager

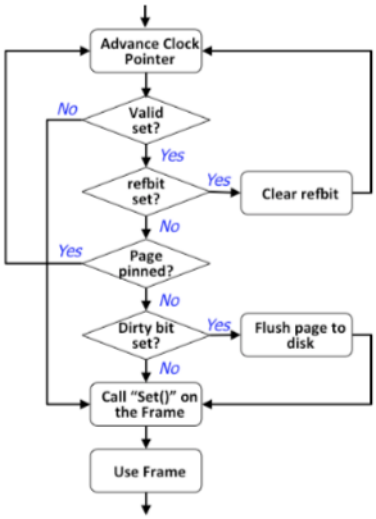

Figure 2: The Clock Replacement Algorithm

住的(pinned)，我们将在下面介绍如何固定页面)。如果被替换的页面是脏的(即该页面被修改过)，则将该页面写回磁盘。图2展示了时钟算法的流程图。

2.2.2 缓冲区管理器的结构

BadgerDB的缓冲区管理器由3个类实现，分别是BufMgr、BufDesc和BufHashTbl。BufMgr类只有一个实例。这个类的主要构成要素是缓冲池，即numBufs个页框构成的数组，每个页框的大小为一个页面的大小。除缓冲区数组外，BufMgr类的实例还包含numBufs个BufDesc类的实例，每个BufDesc类的实例用于描述缓冲池中一个页框的状态。BufMgr类的实例使用一个哈希表来记录当前存储在缓冲池中的页面。该哈希表由BufHashTbl类的实例来实现，该实例是BufMgr类的私有成员变量。这些类的具体描述如下：

BufHashTbl类 BufHashTbl类实现了一个哈希表，用来将文件句柄和页号的组合映射到缓冲池的页框号。该哈希表采用链表法实现。本实验提供了这个类的实现。

```
struct hashBucket {
    File* file; // pointer to a file object (more on this below)
    PageId pageNo; // page number within a f i l e
    FrameId frameNo; // frame number of page in the buffer pool
    hashBucket* next; // next bucket in the chain
};
```

下面是BufHashTbl类的定义。

```
class BufHashTbl
{
private:
```

```
    hashBucket** ht; // pointer to actual hash table
    int HTSIZE;
    int hash(const File* file, const PageId pageNo); //returns a value between 0 and HTSIZE
        -1
public:
    BufHashTbl(const int htSize); // constructor
    ~BufHashTbl(); // destructor

    // insert entry into hash table mapping (file, pageNo) to frameNo
    void insert(const File* file, const int pageNo, const int frameNo);

    // Check if (file, pageNo) is currently in the buffer pool (ie. in
    // the hash table. If so, set the corresponding frame number in frameNo and return true.
    bool lookup(const File* file, const int pageNo, int& frameNo);

    // remove entry obtained by hashing (file, pageNo) from hash table.
    void remove(const File* file, const int pageNo);
};
```

BufDesc类 BufDesc类用于记录缓冲池中每个页框的状态。BufDesc类的所有成员属性都是私有的，并且BufMgr类被声明为它的友元(friend)。尽管看上去有些奇怪，但这种定义方法限制了BufDesc类的私有成员变量只能被BufMgr类访问，而不会被其它类随意修改。BufDesc类的大多数属性的意义都非常明显。

- dirty属性表示页框中的页面是否是脏的(即是否被修改过)。如果dirty属性值为true，则在页框被清空之前，必须将页框中的页面先写回磁盘。
- pinCnt属性记录页框中的页面被引用了多少次(相当于引用计数)。
- refbit属性是时钟算法使用的变量(前面已经介绍过)。
- valid属性记录页框中是否包含了一个有效的页面。

本实验已经提供了BufDesc类的实现，然而你可以根据自己的想法对其进行扩充，使其能够记录额外的信息(如与事务相关的信息)。

```
class BufDesc
{
    friend class BufMgr;
private:
    File* file; // pointer to file object
    PageId pageNo; // page within file
    FrameId frameNo; // buffer pool frame number
    int pinCnt; // number of times this page has been pinned
    bool dirty; // true if dirty; false otherwise
    bool valid; // true if page is valid
    bool refbit; // true if this buffer frame been referenced recently

    void Clear(); // initialize buffer frame
    void Set(File* filePtr, PageId pageNum); // set BufDesc member variable values
    void Print() // print values of member variables
    BufDesc(); // constructor
};
```

BufMgr类 BufMgr类是缓冲区管理器的核心。你需要给出这个类的实现。

```
class BufMgr
{
private:
    FrameId clockHand; // clock hand for clock algorithm
    BufHashTbl *hashTable; // hash table mapping (File, page) to frame number
    BufDesc *bufDescTable; // BufDesc objects, one per frame
    std::uint32_t numBufs; // number of frames in the buffer pool
    BufStats bufStats; // statistics about buffer pool usage

```

```
10      void allocBuf(FrameId & frame); // allocate a free frame using the clock algorithm
11      void advanceClock(); // advance clock to next frame in the buffer pool
12  public:
13      Page *bufPool; // actual buffer pool
14
15      BufMgr(std::uint32_t bufs); // constructor
16      ~BufMgr(); // destructor
17
18      void readPage(File* file, const PageId PageNo, Page*& page);
19      void unPinPage(File* file, const PageId PageNo, const bool dirty);
20      void allocPage(File* file, PageId& PageNo, Page*& page);
21      void disposePage(File* file, const PageId pageNo);
22      void flushFile(const File* file);
23  };
```

这个类的定义如下：

- BufMgr(const int bufs)

  BufMgr类的构造函数。为缓冲池分配一个包含bufs个页面的数组，并为缓冲池的BufDesc表分配内存。当缓冲池的内存被分配后，缓冲池中所有页框的状态被置为初始状态。接下来，将记录缓冲池中当前存储的页面的哈希表被初始化为空。本实验已经实现了该构造函数。

- ~BufMgr()

  BufMgr类的析构函数。将缓冲池中所有脏页写回磁盘，然后释放缓冲池、BufDesc表和哈希表占用的内存。

- void advanceClock()

  顺时针旋转时钟算法中的表针，将其指向缓冲池中下一个页框。

- void allocBuf(FrameId& frame)

  使用时钟算法分配一个空闲页框。如果页框中的页面是脏的，则需要将脏页先写回磁盘。如果缓冲池中所有页框都被固定了(pinned)，则抛出BufferExceededException异常。allocBuf()是一个私有方法，它会被下面介绍的readPage()和allocPage()方法调用。请注意，如果被分配的页框中包含一个有效页面，则必须将该页面从哈希表中删除。最后，分配的页框的编号通过参数frame返回。

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

  首先调用哈希表的lookup()方法检查待读取的页面(file, PageNo)是否已经在缓冲池中。如果该页面已经在缓冲池中，则通过参数page返回指向该页面所在的页框的指针；如果该页面不在缓冲池中，则哈希表的lookup()方法会抛出HashNotFoundException异常。根据lookup()的返回结果，我们处理以下两种情况。

  - 情况1: 页面不在缓冲池中。在这种情况下，调用allocBuf()方法分配一个空闲的页框。然后，调用file->readPage()方法将页面从磁盘读入刚刚分配的空闲页框。接下来，将该页面插入到哈希表中，并调用Set()方法正确设置页框的状态，Set()会将页面的pinCnt置为1。最后，通过参数page返回指向该页框的指针。
  - 情况2: 页面在缓冲池中。在这种情况下，将页框的refbit置为true，并将pinCnt加1。最后，通过参数page返回指向该页框的指针。

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

  将缓冲区中包含(file, PageNo)表示的页面所在的页框的pinCnt值减1。如果参数dirty等于true，则将页框的dirty位置为true。如果pinCnt值已经是0，则抛出PAGENOTPINNED异常。如果该页面不在哈希表中，则什么都不用做。

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

  首先调用file->allocatePage()方法在file文件中分配一个空闲页面，file->allocatePage()返回这个新分配的页面。然后，调用allocBuf()方法在缓冲区中分配一个空闲的页框。接下来，在哈希表中插入一条项目，并调用Set()方法正确设置页框的状态。该方法既通过pageNo参数返回新分配的页面的页号，还通过page参数返回指向缓冲池中包含该页面的页框的指针。

- `void disposePage(File* file, const PageId pageNo)`

  该方法从文件file中删除页号为pageNo的页面。在删除之前，如果该页面在缓冲池中，需要将该页面所在的页框清空并从哈希表中删除该页面。

- `void flushFile(File* file)`

  扫描bufTable，检索缓冲区中所有属于文件file的页面。对每个检索到的页面，进行如下操作：(a) 如果页面是脏的，则调用file->writePage()将页面写回磁盘，并将dirty位置为false；(b) 将页面从哈希表中删除；(c) 调用BufDesc类的Clear()方法将页框的状态进行重置。

  如果文件file的某些页面被固定住(pinned)，则抛出BadBufferException异常。如果检索到文件file的某个无效页，则抛出BadBufferException异常。

3 实验内容

1. 阅读源代码。将源代码压缩文件BufMgr.zip解压，你将得到目录bufmgr。该目录下有如下文件：

   - Makefile: 项目的make文件。你可以通过在shell中运行make来编译项目。
   - main.cpp: 项目的主文件。该文件给出了File类和Page类的使用方法。该文件还给出了一组简单的测试用例，用于测试缓冲区管理器的实现是否正确。你必须增加更多的测试用例来测试程序的正确性。
   - buffer.h: 缓冲区管理器类的定义。无需修改。
   - buffer.cpp: 缓冲区管理器类的实现框架。你需要给出方法的具体实现。
   - bufHash.h: 缓冲池哈希表类的定义。无需修改。
   - bufHash.cpp: 缓冲池哈希表类的实现。无需修改。
   - file.h: 文件类的定义。无需修改。
   - file.cpp: 文件类的实现。无需修改。
   - file_iterator.h: 文件页面迭代器的实现。无需修改。
   - page.h: 页面类的定义。无需修改。
   - page.cpp: 页面类的实现。无需修改。
   - page_iterator.h: 页面记录迭代器的实现。
   - exceptions目录: 所有异常类的实现。如果需要，你可以在此增加更多的文件。

   本实验的源代码包含了丰富的注释。你可以阅读这些注释并了解源代码做了什么以及如何做的。你可以使用Doxygen程序来生成源代码的说明文档，具体方法如下：在bufmgr文件夹下，运行下面的命令生成源代码的说明文档。

   ```
   > make doc
   ```

   这里的>是Linux操作系统的shell命令行提示符，不是命令的一部分。Doxygen生成的源代码说明文档存放在docs文件夹下。你可以使用浏览器打开docs/index.html文件并浏览各个类的属性及方法，以便更好地了解各个类的实现。

2. 使用C++语言实现buffer.cpp文件中的方法。

3. 在main.cpp文件中增加更多的测试用例。

4. 在shell中运行make，编译项目。

5. 在shell中运行./bufmgr来测试程序是否正确。

6. 撰写实验报告。

4 实验要求

1. 本实验由每名学生独立完成。
2. 本实验提供了项目的C++源代码。你的代码风格应当符合C++面向对象程序设计的代码风格，包含定义明确的类及清晰的接口。使用C语言的编程风格将被扣分。
3. 代码须包含Doxygen风格的注释。
4. 使用编译选项`-Wall`打开所有编译警告。实验源代码的Makefile文件中默认使用了`-Wall`编译选项。
5. 善于使用工具。例如，使用`make`编译项目；使用`makedepend`自动生成依赖；使用`perl`、`python`或`bash`编写测试脚本；使用`valgrind`捕捉内存错误；使用`gdb`进行调试；使用`git`进行版本控制。
6. 使用课程实验报告模板撰写实验报告，实验报告以PDF文件提交，文件命名规则为“实验2-学号-姓名-报告.pdf”。
7. 提交源代码时只提交源文件的压缩包（不包含二进制文件），文件命名规则为“实验2-学号-姓名-源代码.zip”。
8. 在进行代码检查时，老师会使用新的测试用例，重新编译你的源代码，并进行测试。因此，请不要修改现有代码的接口。如果你修改了接口并因此导致程序无法编译，则你会被扣分。

5 实验评价

本实验的成绩构成如下:

- 程序正确性: 60%
- 代码风格: 10%
- 代码讲解: 10%
- 实验报告: 20%