1. **实验目标**

本次实验要实现一个页面的缓冲区管理器。具体要实现以下的函数：

~BufMgr()：

清除所有脏页并释放缓冲池和BufDesc表

void advanceClock()：

用来找到下一个时钟的位置

void allocBuf(FrameId& frame)：

使用时钟算法分配自由帧；如有必要，将脏页写回磁盘。

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

通过调用lookup（）方法检查页是否已经在缓冲池中。当页不在缓冲池中可以在哈希表上抛出HashNotFoundException以获取帧编号。

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

减少一个页面的占用次数

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

通过调用file->allocatePage（）方法在指定的文件中分配一个空页。此方法将返回新分配的页。然后调用allocBuf（）以获取缓冲池帧。接下来，将一个条目插入到哈希表中，并在帧上调用Set（），以正确设置它。

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

功能是释放一个页面

void flushFile(File\* file)

功能是找到含有对应文件的页面，并释放

1. **代码实现**

BufMgr::~BufMgr(){  
 delete hashTable;  
 delete[] bufPool;  
 delete[] bufDescTable;  
}

直接调用delete删除哈希表、缓冲池、缓冲池表

void BufMgr::advanceClock(){  
 clockHand++;  
 if (clockHand >= numBufs){  
 clockHand %= numBufs;  
 }  
}

将时钟提前到缓冲池的下一帧。

如果指针超过了最大值，进行取模操作。

void BufMgr::allocBuf(FrameId &frame){  
 unsigned pinned = 0;  
 while (true){  
 advanceClock();  
 if (!bufDescTable[clockHand].valid){  
 frame = clockHand;  
 return;  
 }  
 if (bufDescTable[clockHand].refbit){  
 bufDescTable[clockHand].refbit = false;  
 continue;  
 }  
 if (bufDescTable[clockHand].pinCnt){  
 pinned++;  
 if (pinned == numBufs){  
 throw BufferExceededException();  
 }else{  
 continue;  
 }  
 }  
 if (bufDescTable[clockHand].dirty){  
 bufDescTable[clockHand].file->writePage(bufPool[clockHand]);  
 bufDescTable[clockHand].dirty = false;  
 }  
 frame = clockHand;  
 if (bufDescTable[clockHand].valid){  
 try{  
 hashTable->remove(bufDescTable[clockHand].file, bufDescTable[clockHand].pageNo);  
 }  
 catch (HashNotFoundException &){  
 }  
 }  
 break;  
 }  
}

遍历栈区寻找可用的页面。如果是没有被使用过的页面，直接进行分配。如果缓冲区所有的页面都被占用，那么会进行报错BufferExceededException()。如果找到脏页，会将它写回磁盘，并将脏页标记给清除。如果不是脏页，那么就进行分配操作。如果它在哈希表中要将它移除。

void BufMgr::readPage(File \*file, const PageId pageNo, Page \*&page){  
 FrameId frame;  
 try{  
 hashTable->lookup(file, pageNo, frame);  
 bufDescTable[frame].refbit = true;  
 bufDescTable[frame].pinCnt++;  
 page = (bufPool + frame);  
 }catch (HashNotFoundException &){  
 allocBuf(frame);  
 bufPool[frame] = file->readPage(pageNo);  
 hashTable->insert(file, pageNo, frame);  
 bufDescTable[frame].Set(file, pageNo);  
 page = (bufPool + frame);  
 }  
}

如果页面在缓冲池中，增加它的占用次数，调用page返回指向该页面的指针

如果页面不在缓冲池中，那么将页面读取到缓冲池，插入哈希表中，调用set正确设置该界面，调用page返回指向该页面的指针

void BufMgr::unPinPage(File \*file, const PageId pageNo, const bool dirty){  
 FrameId frame;  
 try{  
 hashTable->lookup(file, pageNo, frame);  
 }catch (HashNotFoundException &){  
 //没有该页面  
 cerr << "Warning: unpinning a nonexistent page" << endl;  
 return;  
 }  
 //找到页面  
 if (bufDescTable[frame].pinCnt > 0){  
 bufDescTable[frame].pinCnt--;  
 if (dirty){  
 bufDescTable[frame].dirty = true;  
 }  
 }else{  
 //pin = 0,抛出异常  
 throw PageNotPinnedException(bufDescTable[frame].file->filename(), bufDescTable[frame].pageNo, frame);  
 }  
}

如果缓冲池中没有该页面，进行异常提示。

如果在缓冲池中，那么将它的占用次数减少。如果占用次数为0，进行报错。

void BufMgr::flushFile(const File \*file){  
 for (FrameId fi = 0; fi < numBufs; fi++){  
 if (bufDescTable[fi].file == file){  
 if (!bufDescTable[fi].valid){  
 throw BadBufferException(fi, bufDescTable[fi].dirty, bufDescTable[fi].valid, bufDescTable[fi].refbit);  
 }  
 if (bufDescTable[fi].pinCnt > 0){  
 throw PagePinnedException(file->filename(), bufDescTable[fi].pageNo, fi);  
 }  
 if (bufDescTable[fi].dirty){  
 bufDescTable[fi].file->writePage(bufPool[fi]);  
 bufDescTable[fi].dirty = false;  
 }  
 hashTable->remove(file, bufDescTable[fi].pageNo);  
 bufDescTable[fi].Clear();  
 }  
 }  
}

遍历整个表，找到含有对应页面的缓冲页，移除并清空该页面。如果页面是脏页，则将其写回磁盘，初始化脏页标记。如果页面被占用或者页面不可用，则进行报错。

void BufMgr::allocPage(File \*file, PageId &pageNo, Page \*&page){  
 FrameId frame;  
 Page p = file->allocatePage();  
 allocBuf(frame);  
 bufPool[frame] = p;  
 pageNo = p.page\_number();  
 hashTable->insert(file, pageNo, frame);  
 bufDescTable[frame].Set(file, pageNo);  
 page = bufPool + frame;  
}

掉用allocatePage()分配一个新页面，加入哈希表，调用set()，返回该页面指针。

void BufMgr::disposePage(File \*file, const PageId PageNo){  
 FrameId frame;  
 try{  
 hashTable->lookup(file, PageNo, frame);  
 hashTable->remove(file, PageNo);  
 bufDescTable[frame].Clear();  
 }catch (HashNotFoundException &){  
 }  
 file->deletePage(PageNo);  
}

删除一个页面。如果它在缓冲池中，要将缓冲内容一并删除。

**三、实验结果**

12个样例均能通过，实验结果如下：

