Buffer Manager

姓名：王丁子睿 学号：1183710211

1. 实验过程

以下将按顺序介绍各个方法的实现

* 1. advanceClock

时针法模拟FIFO时，clockHand指针往后移一位。

    void BufMgr::advanceClock()

    {

        // increase clock hand

        clockHand = (clockHand + 1) % numBufs;

    }

* 1. allocBuf

为一个新的帧分配空间，依顺序判断：

* 当前位置是否可用。
* 当前位置是否为其他帧占用，若所有位置都被占用，则报错。
* 当前位置的refbit是否为假（用于实现时针法）。

若以上判断均通过，则进入替换流程：

* 若该位置dirty，则写回。
* 令当前帧指向该位置（分配给该帧）。
* 清除当前位置的原有信息。

    void BufMgr::allocBuf(FrameId &frame)

    {

        FrameId lastClockHand = clockHand;

        unsigned int pinnedCount = 0;

        while (true)

        {

            advanceClock();

            // find available position

            if (!bufDescTable[clockHand].valid)

            {

                frame = clockHand;

                return;

            }

            if (bufDescTable[clockHand].pinCnt > 0)

            {

                pinnedCount += 1;

                if (pinnedCount == numBufs)

                {

                    throw BufferExceededException();

                }

                continue;

            }

            if (bufDescTable[clockHand].refbit)

            {

                bufDescTable[clockHand].refbit = false;

                continue;

            }

            // allocate position

            if (bufDescTable[clockHand].dirty)

            {

                bufDescTable[clockHand].file->writePage(bufPool[clockHand]);

                bufDescTable[clockHand].dirty = false;

            }

            frame = clockHand;

            try

            {

                hashTable->remove(bufDescTable[clockHand].file, bufDescTable[clockHand].pageNo);

            }

            catch (HashNotFoundException e)

            {

                // handle hash not found exception

                return;

            }

            if (lastClockHand == clockHand)

                throw BufferExceededException();

            break;

        }

    }

* 1. readPage

从buffer中读取指定页。

若buffer中存在该页，则直接读取；否则，先分配空间再读取。

    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 e)

        {

            allocBuf(frame);

            bufPool[frame] = file->readPage(pageNo);

            hashTable->insert(file, pageNo, frame);

            bufDescTable[frame].Set(file, pageNo);

            page = &bufPool[frame];

        }

    }

* 1. unPinPage

设置某页不再占用某个空间。

同时若该页dirty，则该空间的信息需要写回。

    void BufMgr::unPinPage(File \*file, const PageId pageNo, const bool dirty)

    {

        FrameId frame;

        try

        {

            hashTable->lookup(file, pageNo, frame);

        }

        catch (HashNotFoundException e)

        {

            // handle hash not found exception

            return;

        }

        if (bufDescTable[frame].pinCnt > 0)

        {

            bufDescTable[frame].pinCnt--;

            if (dirty)

            {

                bufDescTable[frame].dirty = true;

            }

        }

        else

        {

            throw PageNotPinnedException(bufDescTable[frame].file->filename(), bufDescTable[frame].pageNo, frame);

        }

    }

* 1. flushFile

清除信息，需要满足页可用且引用数（pinCnt）为零。

    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();

            }

        }

    }

* 1. allocPage

为某个页分配空间，流程为先分配一个可用空间，然后给这块空间设置相应信息。

    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];

    }

* 1. disposePage

删除一个页。

    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 e)

        {

            //not in the table; do nothing

        }

        file->deletePage(PageNo);

    }

1. 实验结果

文本

描述已自动生成

1. 实验心得

* 加深了对缓冲区管理的理解。
* 提高了C++面向对象编程的能力。