哈尔滨工业大学

**<<数据库系统>>**

**实验报告三**

**(2024年度秋季学期)**

|  |  |
| --- | --- |
| **姓名：** |  |
| **学号：** |  |
| **学院：** | **计算学部** |
| **教师：** | **张浩** |

实验三

## 实验目的

实验的主要目的是通过实现一个缓冲管理器，帮助理解关系数据库管理系统（RDBMS）中的关键组件。本实验的任务是基于已有的存储管理器，开发一个缓冲管理器。具体目标包括：

1. 理解缓冲池的结构和管理机制。
2. 掌握缓冲替换策略（如时钟算法）的实现原理。
3. 深入了解文件和页面操作的抽象（I/O层）。
4. 应用C++面向对象的编程方法进行设计和实现。

## 实验环境

1. **操作系统**：Linux（实验中明确提到在Linux环境下运行）。语言：en-us，防止中文界面的显示乱码问题。
2. **开发工具**：C++编译器（Makefile 中启用 -Wall 参数以激活所有警告），推荐工具包括 valgrind（内存调试）、gdb（调试器）、git（版本控制）等。
3. **项目结构**：通过提供的 BufMgr.zip 包进行开发，解压后包含必要的头文件、实现文件以及测试驱动程序。
4. **运行指令**：主要使用 make 构建项目，生成文档命令为 make doc。

## 实验过程及结果

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。

**3.1 BadgerDB 的 I/O 层**

BadgerDB数据库系统的最底层是 I/O 层，该层为系统的上层提供文件和页面管理的基本操作功能，具体包括：

**1. 功能描述**

文件操作：支持文件的创建与销毁。提供打开和关闭文件的接口。

页面操作：在文件中分配和释放页面。从文件中读取页面或将数据写入页面。

**2. 核心组件**

I/O 层由以下两个主要类构成：

File 类：提供对文件的抽象。通过方法实现文件的读写操作。

Page 类：提供对页面的抽象。支持对页面内容的操作。

**3.2 BadgerDB**的缓冲区管理器

数据库缓冲池（buffer pool）由一组固定大小的内存缓冲区（buffer）组成，它是用于存储从磁盘加载到内存中的数据库页面（page，也称为磁盘块）的一个数组。在缓冲池中，每个固定大小的缓冲区称为页框（frame）。页面是磁盘与缓冲池之间进行数据传输的基本单位。现代数据库管理系统通常使用至少8KB的页面大小。

当页面首次从磁盘加载到缓冲池时，缓冲池中的页面内容与磁盘上的对应页面完全一致。然而，如果数据库管理系统（DBMS）对缓冲池中的页面进行了修改，那么缓冲池中的页面就会与磁盘上的原始页面不同。这样的页面被称为“脏页面”。

由于磁盘上的数据库通常比缓冲池的大小大得多，任何时候缓冲池中只能存储一部分页面。缓冲区管理器（buffer manager）的作用是负责管理哪些页面可以保留在缓冲池中。当缓冲区管理器接收到页面访问请求时，它会先检查所需页面是否已经在缓冲池中。如果页面已经存在，缓冲区管理器会直接返回指向该页框的指针。如果页面不在缓冲池中，缓冲区管理器将释放一个页框（如果需要释放的页框包含脏页面，则会先将该页面写回磁盘），然后将所需页面从磁盘加载到刚释放的页框中。

**3.2.1 缓冲区页面替换策略**

当需要在缓冲池中获取一个空闲页框时，需要决定从缓冲池中替换掉哪个页面。常见的页面替换策略包括 FIFO（先进先出）、MRU（最近最常使用）和 LRU（最近最少使用）。虽然 LRU 是广泛使用的一种策略，但由于其实现开销较大，在数据库系统的许多常见运行场景中并不是最优选择。

为了在性能和实现复杂度之间取得平衡，许多系统选择采用 时钟算法（Clock Algorithm），它通过近似 LRU 的行为，在效率和效果之间实现了良好的折衷。

**1. 时钟算法的基本原理**

时钟算法将缓冲池中的页框组织成一个环形队列，每个页框都包含一个引用位（refbit），用来指示页面最近是否被访问过。算法的核心是一个指针（表针），指向缓冲池中的某个页框。

**环形布局：**

假设缓冲池包含 numBufs 个页框，页框编号为 0 到 numBufs - 1。所有页框在概念上形成一个环形结构。表针通过一个整数值表示，范围为 0 到 numBufs - 1。

**引用位的使用：**

当某个页面被访问时（例如通过调用缓冲区管理器的 readPage() 方法），该页面所在页框的引用位被设置为 1。引用位的初始值为 0，表示页面较久未被访问。

**表针的移动：**

表针顺时针移动，每次指向一个页框。表针移动的实现是通过对页框编号加 1 后对 numBufs 取模。

**2. 页面替换过程**

每当需要为新页面分配页框时，时钟算法执行以下步骤：

**检查引用位：**

当表针指向某个页框时，检查该页框的引用位。若引用位为 1，说明该页框中的页面最近被访问过：将引用位清零。表针移动到下一个页框。若引用位为 0，说明该页面较久未被访问，可以被替换。

**页面替换：**

如果找到的页框中页面未被固定（unpinned）：若该页面是脏页面，则需将其写回磁盘。清空页框，并加载新页面。若页框中的页面被固定（pinned），表针继续移动。

**3. 算法特点与优势**

效率高：相比 LRU，时钟算法避免了复杂的链表操作，仅通过引用位和表针即可实现替换策略。

近似 LRU 行为：通过引用位的使用，确保最近被访问的页面不容易被替换。

简单实现：基于环形结构，算法逻辑简单且易于维护。

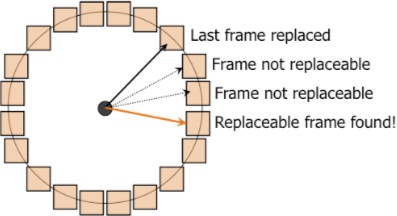


图 1：缓冲池的环形布局示意图  
每个正方形表示一个页框，包含页面数据和引用位。

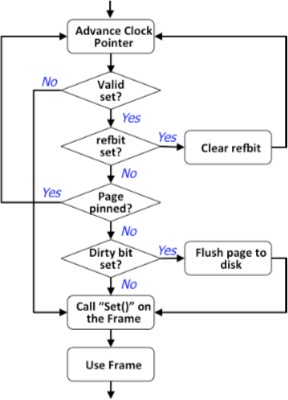


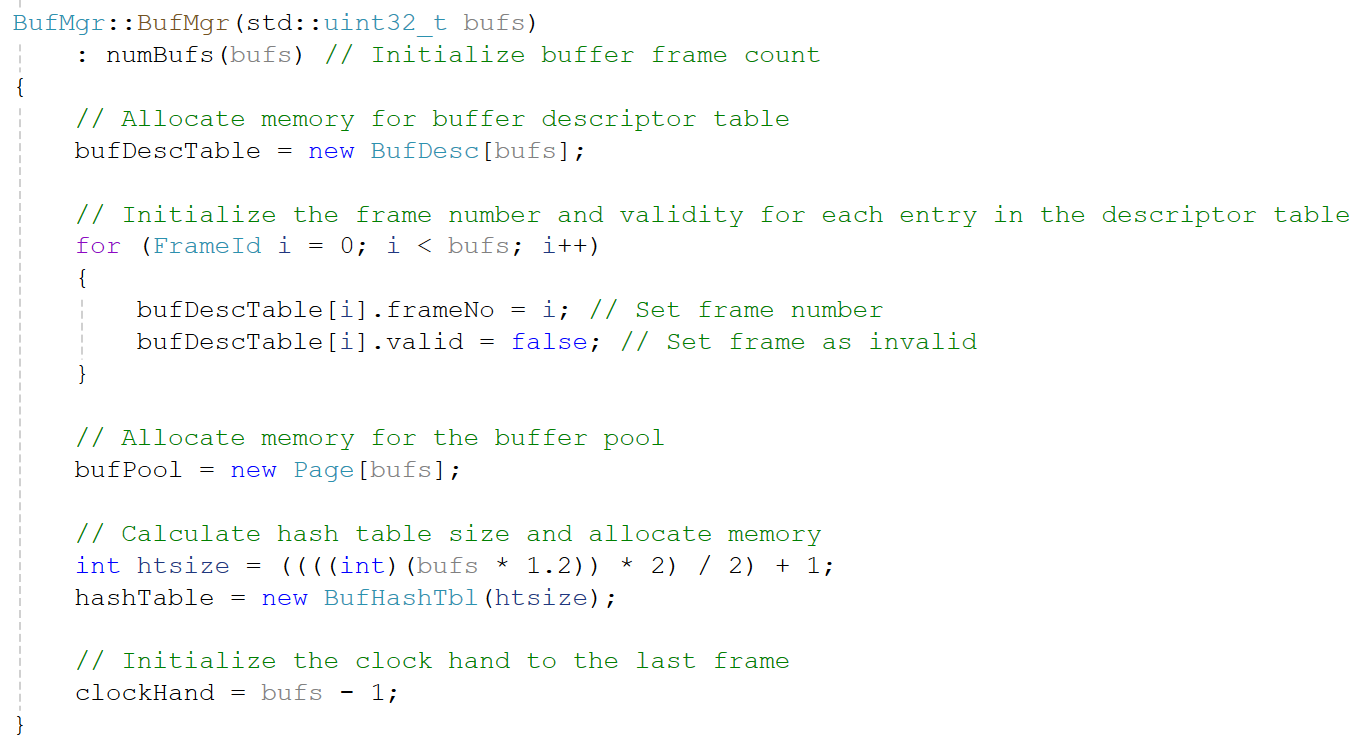
图 2：时钟替换算法流程图

**3.2.2 缓冲区管理器的结构**

使用三个 C++ 类： BufMgr、BufDesc 和 BufHashTbl。 BufMgr 类只有一个实例。该类的一个关键组件是实际的缓冲池，它由 numBufs 框架数组组成，每个框架的大小相当于一个数据库页面。除了这个数组外，BufMgr 实例还包含一个 BufDesc 类的 numBufs 实例数组，用于描述缓冲池中每个帧的状态。哈希表用于跟踪当前驻留在缓冲池中的页面。这个哈希表由 BufHashTbl 类的一个实例实现。该实例是 BufMgr 类的私有数据成员。下文将详细介绍这些类。

• 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。释放缓冲区管理器的资源。

文本

描述已自动生成

• void advanceClock()

时钟算法中的表针，将其指向缓冲池中下一个页框。

图形用户界面, 文本

描述已自动生成

• void allocBuf(FrameId& frame)

使用时钟算法分配空闲帧；必要时，将脏页写回磁盘。如果所有缓冲区框架都被钉住，则抛出 BufferExceededException。该私有方法将被下面描述的 readPage() 和 allocPage() 方法调用。确保如果分配的缓冲帧中有有效页面，则从哈希表中删除相应的条目。文本

描述已自动生成

图形用户界面, 文本

描述已自动生成

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

首先，通过调用哈希表上的 lookup() 方法（当页面不在缓冲池中时可能会抛出 HashNotFoundException）检查页面是否已在缓冲池中，以获取帧号。根据 lookup() 调用的结果，有两种情况需要处理：

- 情况 1：页面不在缓冲池中。调用 allocBuf() 分配一个缓冲区框架，然后调用 file->readPage() 方法将页面从磁盘读入缓冲池框架。然后，将页面插入哈希表。最后，在框架上调用 Set() 方法对其进行正确设置。Set() 会将页面的 pinCnt 设置为 1。通过页面参数返回包含页面的框架指针。

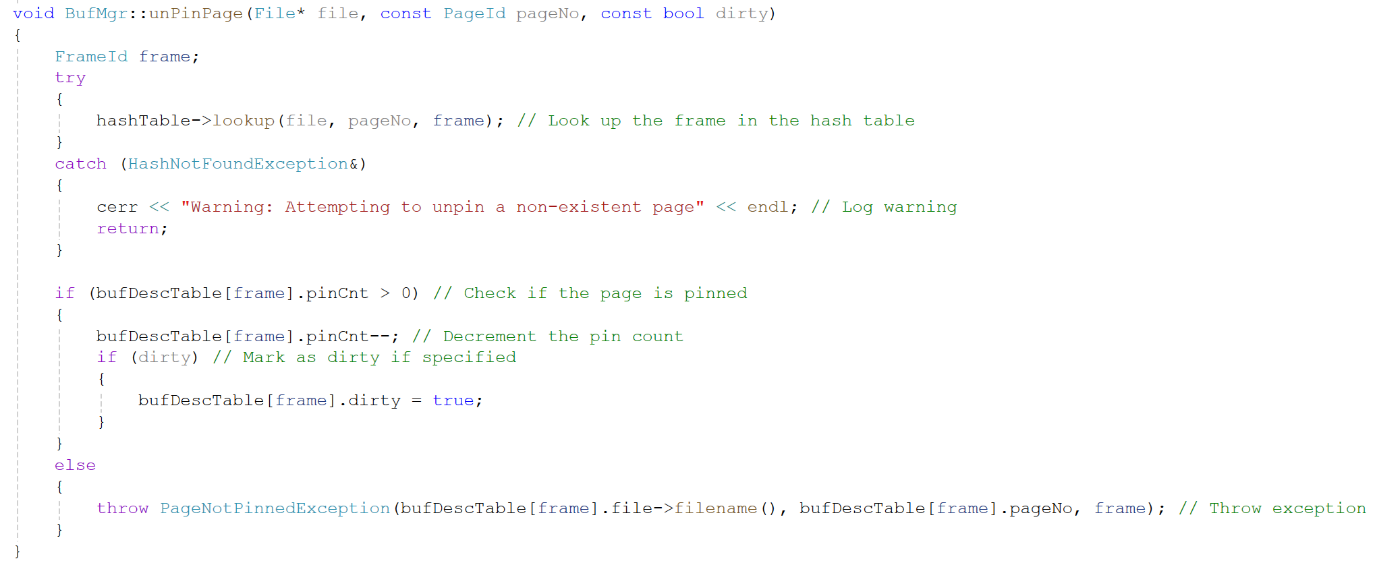
- 情况 2：页面位于缓冲池中。在这种情况下，设置相应的 refbit、递增页面的 pinCnt

然后通过页面参数返回指向包含页面的帧的指针。

文本

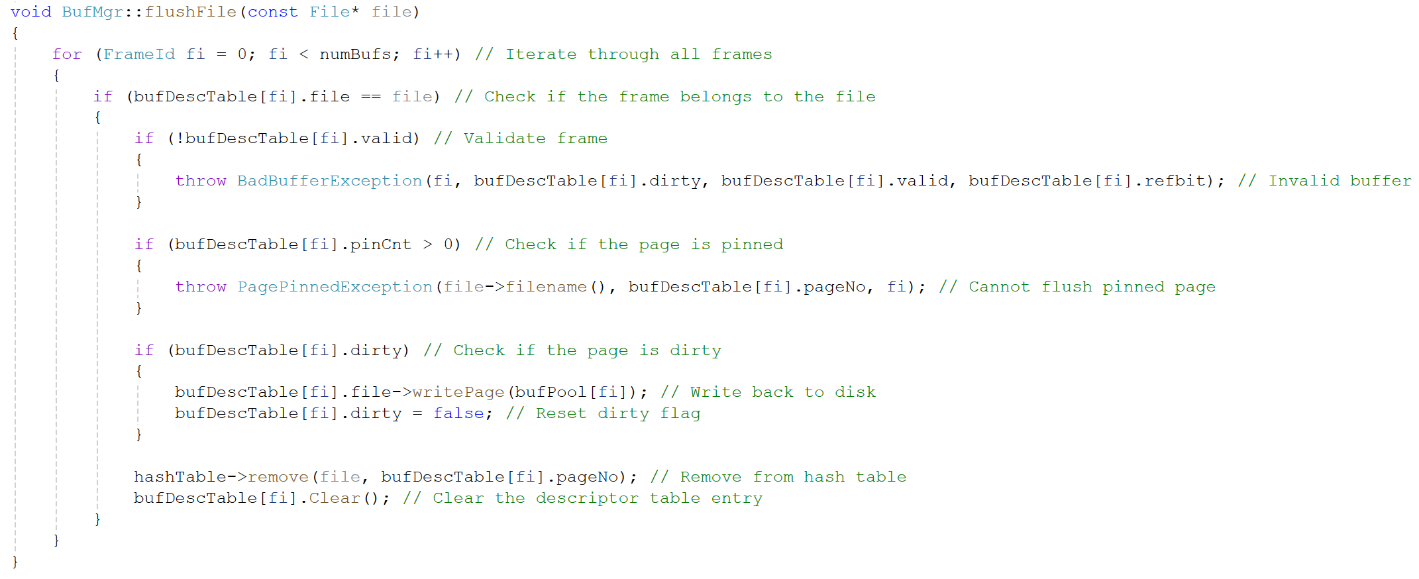
描述已自动生成

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

减少包含（file, PageNo）的帧的针脚数，如果 dirty == true，则设置 dirty 位。如果针脚数已为 0，则抛出 PAGENOTPINNED。 如果在哈希表中未找到页面，则不做任何操作。

• void flushFile(File\* file)

应扫描 bufTable 以查找属于文件的页面。对于遇到的每个页面，它应该 (a) 如果页面是脏的，则调用 file->writePage() 将页面刷新到磁盘，然后将页面的脏位设置为 false，(b) 从 hashtable 中删除页面（无论页面是干净的还是脏的），(c) 为页面帧调用 BufDesc 的 Clear() 方法。

如果文件的某个页面被钉住，则抛出 PagePinnedException。如果遇到属于文件的无效页面，则抛出 BadBufferException。

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

该方法的第一步是通过调用 file->allocatePage() 方法在指定文件中分配一个空页面。该方法将返回一个新分配的页面。然后调用 allocBuf() 获得一个缓冲池框架。然后，在哈希表中插入一个条目，并调用 Set() 方法对框架进行正确设置。该方法通过 pageNo 参数向调用者返回新分配页面的页码，并通过 page 参数返回为该页面分配的缓冲区框架指针。文本

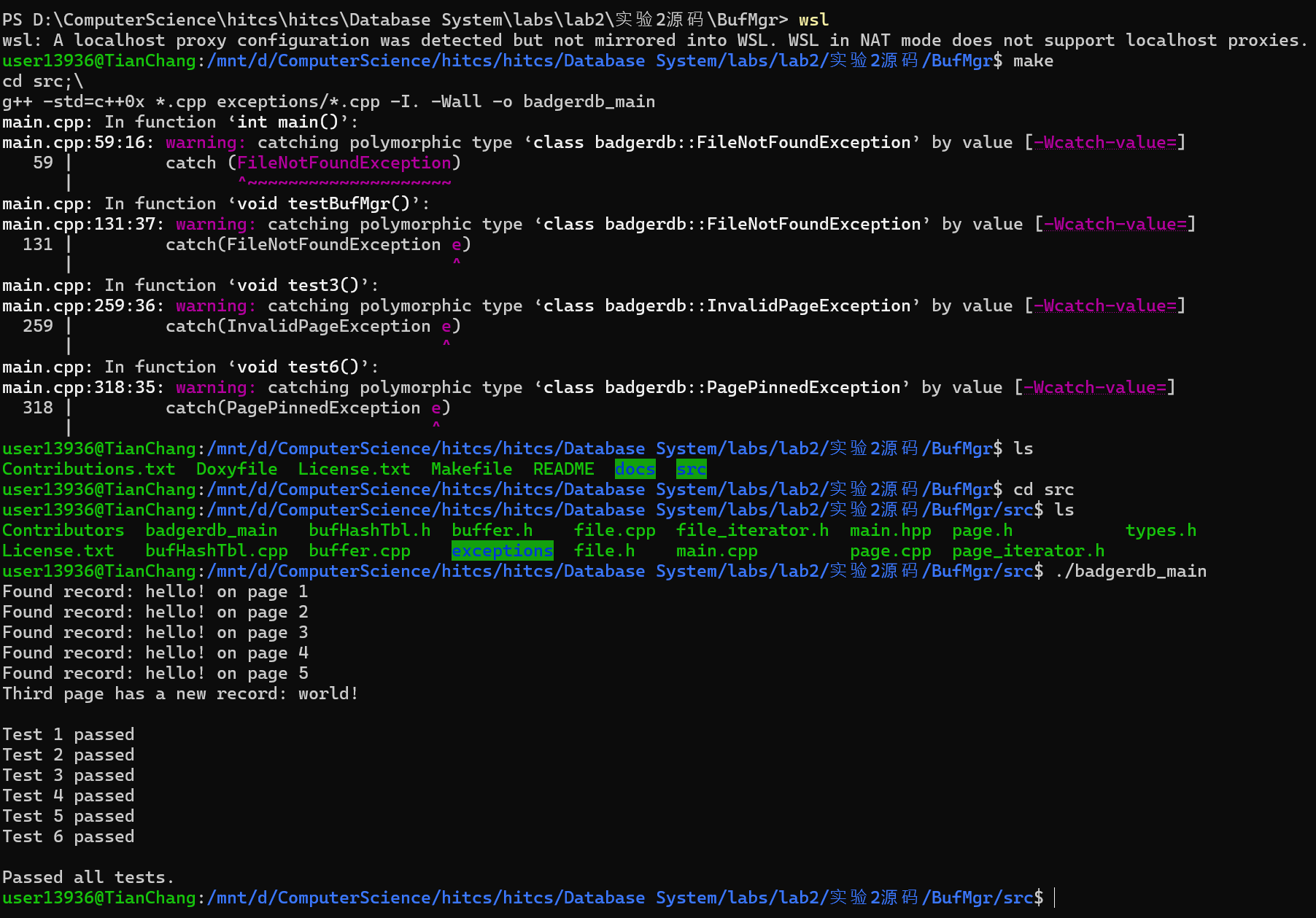
描述已自动生成• void disposePage(File\* file, const PageId pageNo)

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。

文本

描述已自动生成

实验结果如下：



## 实验心得

通过这次数据库实验，我对缓冲区管理器的工作原理和实现细节有了更深刻的理解。特别是在实现过程中，时钟算法的设计和应用让我对页面替换策略的选择及其对系统性能的影响有了直观的认识。相比于理论学习，亲自动手编写代码并进行调试的过程让我真正体会到算法在实际系统中的实现复杂性和优化的重要性。