
高级数据库系统课程实验

Storage and Buffer Manager

实验报告

姓 名: 李 爽

学 号: SA19225033

班 级: 软设四班

实验日期: 2020 年 7 月 7 日

一、实验背景

为了加深对数据库课程学习的理解，进行了此实验，实现了一个简单的存储和缓存管理器。在这个实验中涉及了缓存技术、应用 Hash 散列技术组织缓冲区、文件的存储结构等各个方面的知识。

二、实验环境

开发环境：操作系统：Windows 10（64）

开发语言：C++

集成开发环境：Visual Studio 2017，Windows SDK 10.0.17134.0

三、实验内容

- 1、实现对此磁盘中的数据进行读写。
- 2、实现应用 Hash 散列技术及双向链表组织缓冲区。
- 3、使用文件老师所给数据文件验证系统。
- 4、更改缓存器大小并对产生现象进行分析。

三、设计思路与实现

1、实现对此磁盘中的数据进行读写。

文件存储格式如下：

每条记录长度为 316 字节

页面大小设定为 4KB

则每个磁盘页可放入 12 条记录

页面数为 50000

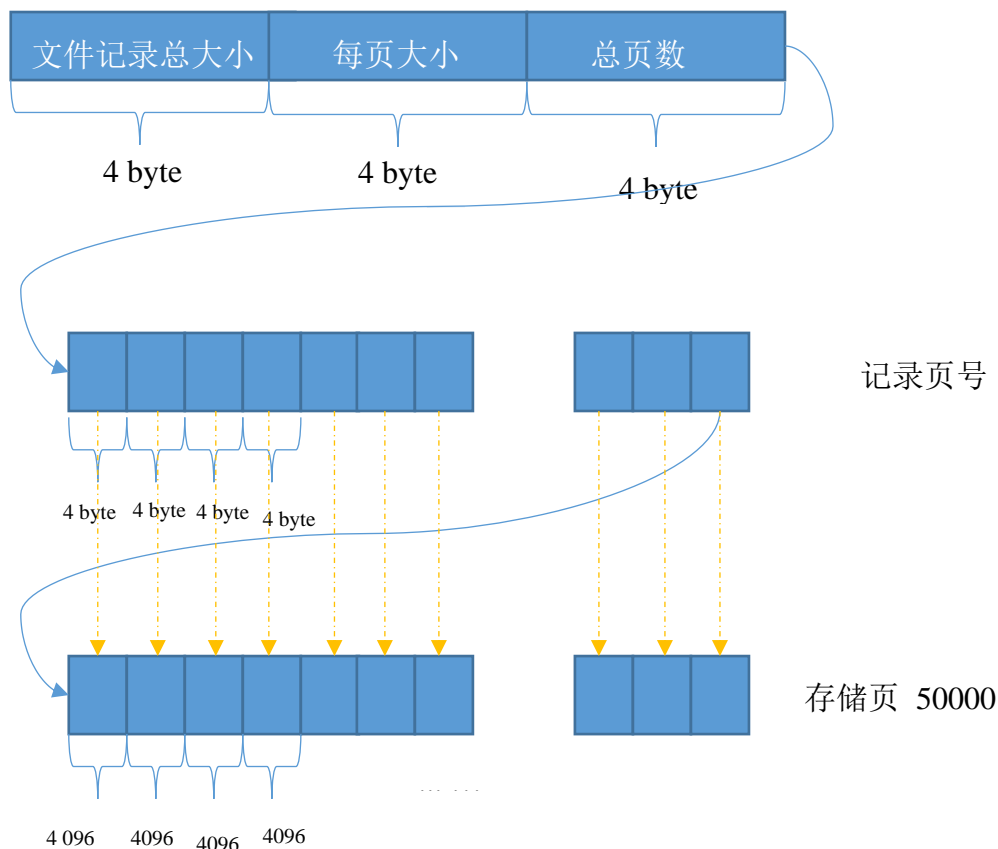


图 1.1 分页存储设计

```

for (int j = 0; j < num_blocks; j++)
// 写入num_blocks个块，数据记录方法：固定格式定长记录
{
// 写入块首部
block_num = j;
int writeCnt = fwrite(&block_num, sizeof(block_num), 1, fstream_w); // 数据块号
writeCnt = fwrite(&timestamp, sizeof(timestamp), 1, fstream_w); // 时间戳
writeCnt = fwrite(&offset, sizeof(offset), 1, fstream_w); // 块内偏移表
// 写入记录
char * p_schema = NULL;
unsigned int timestamp = j;
unsigned int length = 316;
char namePtr[32];
char addressPtr[256];
char genderPtr[4];
char birthdayPtr[12];
for (int i = 0; i < FRAMESIZE / RECORDSIZE; i++) // 2048/316=12
{
writeCnt = fwrite(&fstream_w, sizeof(fstream_w), 1, fstream_w);
writeCnt = fwrite(&length, sizeof(length), 1, fstream_w);
writeCnt = fwrite(&timestamp, sizeof(timestamp), 1, fstream_w);
writeCnt = fwrite(&namePtr, sizeof(namePtr), 1, fstream_w);
writeCnt = fwrite(&addressPtr, sizeof(addressPtr), 1, fstream_w);
writeCnt = fwrite(&genderPtr, sizeof(genderPtr), 1, fstream_w);
writeCnt = fwrite(&birthdayPtr, sizeof(birthdayPtr), 1, fstream_w);
}
char null[272]; // 填充字节使得4KB对齐
writeCnt = fwrite(&null, sizeof(null), 1, fstream_w);
}

```

图 1.2 数据库文件建立函数

```
void NoIndexQuery(int find_block_num)
{
    // 查询指定块的块号
    FILE * fstream_r;
    fstream_r = fopen("data.dbf", "rb");
    unsigned int blocknum = 0;
    cout << "query target: " << find_block_num << endl;
    int offset = find_block_num * FRAMESIZE;
    fseek(fstream_r, offset, SEEK_SET);
    int readCnt = fread(&blocknum, sizeof(blocknum), 1, fstream_r);
    cout << " query result: " << blocknum << endl;

    fclose(fstream_r);
}
```

图 1.3 数据库文件访问测试

首先建立数据库，名称为 data.dbf。数据库只需建立一次，得到数据库文件后可以将 CreateBlockWRTest(int num_blocks)函数的调用注释掉，并备份测试数据库。

2、缓存管理器设计

2.1 确定缓存区的结构

缓存页大小设为 4KB。

缓存区中 frame 的数量初始定为 1024。

2.2 缓存控制块设计（BCB）

一般地，1 个 BCB 维护着 1 个磁盘页在缓存（内存）中的信息，包括磁盘页号、此块号对应的缓存页号、用户占用计数、时间戳、脏位。BCB 结构体如图 2.2。在主流的实现方式中，BCB 作为数组或者链表的结点，组成便于管理的数据结构，提供快速的查找、更新、置换等功能。

```

struct BCB
{
    int page_id, frame_id, ref, count, time, dirty;
    BCB * _prev;
    BCB * _next;
    BCB() : _next(NULL), _prev(NULL) {}
    BCB(int page_id, int frame_id) : page_id(page_id),
    frame_id(frame_id), count(0), time(0), dirty(0) { BCB(); }
    // 断开本节点
    void disconnect() {
        if (_prev) _prev->_next = _next;
        if (_next) _next->_prev = _prev;
    }
    // next方向插入节点
    void Insert(BCB * node) {
        if (_next) {
            node->_prev = this; // this
            _next->_prev = node;
            node->_next = _next;
            _next = node;
        }
    }
};

```

图 2.2 BCB 结构设计

2.3 基于 Hasu 与双向链表的缓存区

LRUCache 的 private 成员:

```

class LRUCache {
private:
    int size, capacity; //当前大小, 容量上限
    unordered_map<int, BCB*> hashmap;
    stack<int> frame_ids;
    BCB * head, *tail;
    int hitNum; // 命中数
    int IONum; // 磁盘IO
    bBuff * buff;
    int f2p[BUFFSIZE]; // frame_id到page_id的映射
    DiskSpaceManager diskSpaceManager;

```

LRUCache 的 public 成员:

```

public:
    LRUCache(int capacity, char dbpath[]) : size(0), capacity(capacity)
    // 读出LRUCache对应的frame_id
    int read(int page_id)
    // 仅查看frame_id
    int read_id(int page_id)
    // 写入LRUCache对应的frame_id
    int put(int page_id, int frame_id = 0)
    // 查看要换出的frame_id
    int selectVictim()
    // 从缓存删除
    void remove(int page_id)
    // 脏位
    int setDirty(int page_id)
    // 用户数
    int setCount(int page_id)
    // 完成对象LRUCache释放前的一些工作
    void saveDirty2Disk()

```

2.4 实现磁盘文件操作

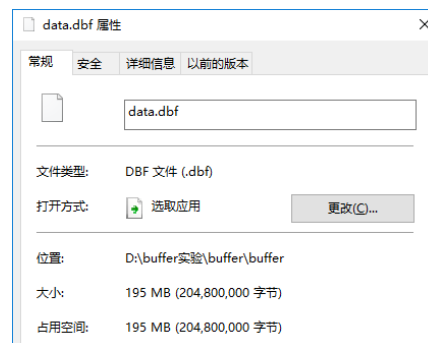
文件的流操作：

```
class DiskSpaceManager {
private:
    FILE * fstream;
public:
    DiskSpaceManager() :fstream(NULL) {}
    void openFile(char filepath[])
    void readDisk(int page_id, int frame_id, bBuff * buff)
    void writeDisk(int page_id, int frame_id, bBuff * buff)
};
```

五、实验结果及分析

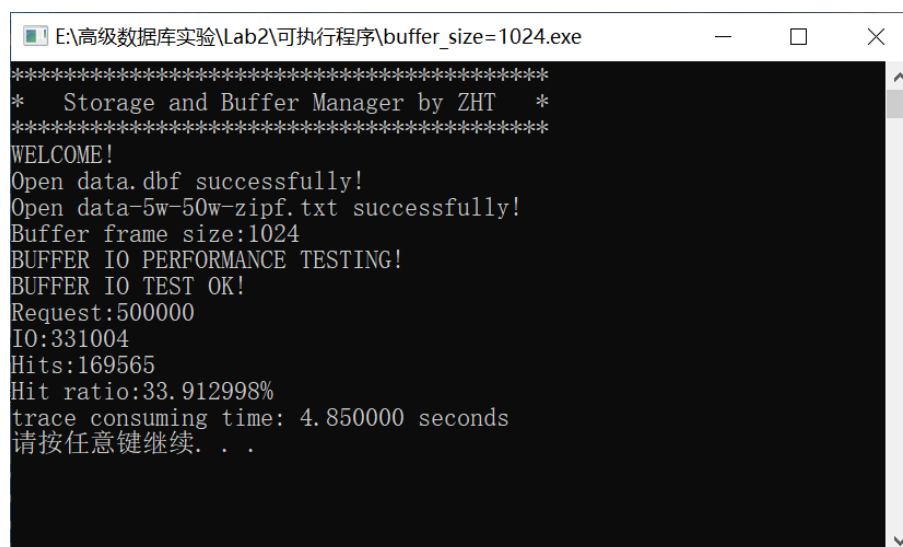
1、生成数据库文件

使用 CreateBlockWRTest 函数生成含 50000 个磁盘页的数据库文件 data.dbf



2、对不同缓存区大小进行实验

① 设置 BUFFSIZE=1024 进行测试，结果如下图：

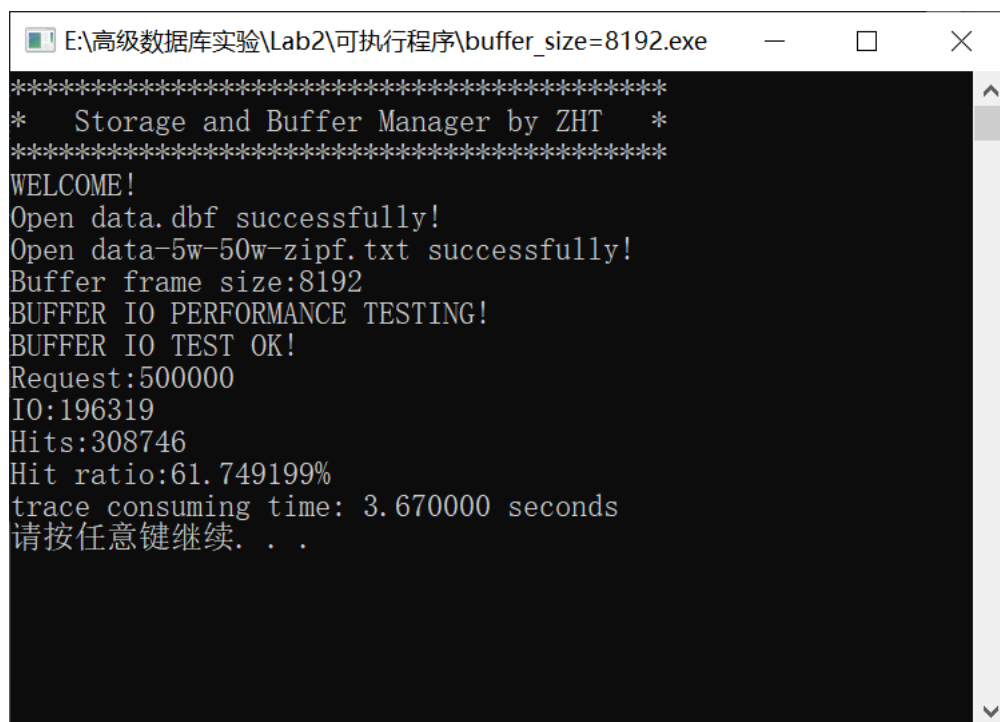


②设置 BUFFSIZE=4096 进行测试，结果如下图：



```
*****
*   Storage and Buffer Manager by ZHT   *
*****
WELCOME!
Open data.dbf successfully!
Open data-5w-50w-zipf.txt successfully!
Buffer frame size:4096
BUFFER IO PERFORMANCE TESTING!
BUFFER IO TEST OK!
Request:500000
IO:246942
Hits:255440
Hit ratio:51.088001%
trace consuming time: 4.179000 seconds
请按任意键继续. . .
```

③设置 BUFFSIZE=8192 进行测试，结果如下图：



```
*****
*   Storage and Buffer Manager by ZHT   *
*****
WELCOME!
Open data.dbf successfully!
Open data-5w-50w-zipf.txt successfully!
Buffer frame size:8192
BUFFER IO PERFORMANCE TESTING!
BUFFER IO TEST OK!
Request:500000
IO:196319
Hits:308746
Hit ratio:61.749199%
trace consuming time: 3.670000 seconds
请按任意键继续. . .
```

④设置 BUFFSIZE=16384 进行测试，结果如下图：

```
E:\高级数据库实验\Lab2\可执行程序\buffer_size=1638...
*****
*   Storage and Buffer Manager by ZHT   *
*****
WELCOME!
Open data.dbf successfully!
Open data-5w-50w-zipf.txt successfully!
Buffer frame size:16384
BUFFER IO PERFORMANCE TESTING!
BUFFER IO TEST OK!
Request:500000
IO:141795
Hits:369294
Hit ratio:73.858803%
trace consuming time: 3.177000 seconds
请按任意键继续. . .
```

⑤ 设置 BUFFSIZE=32768 进行测试，结果如下图：

```
E:\高级数据库实验\Lab2\可执行程序\buffer_siz...
*****
*   Storage and Buffer Manager by ZHT   *
*****
WELCOME!
Open data.dbf successfully!
Open data-5w-50w-zipf.txt successfully!
Buffer frame size:32768
BUFFER IO PERFORMANCE TESTING!
BUFFER IO TEST OK!
Request:500000
IO:93589
Hits:432375
Hit ratio:86.474998%
trace consuming time: 2.402000 seconds
请按任意键继续. . .
```

| BUFFSIZE | IO | Hits | Hit ratio | time/s | memory/MB |
|----------|--------|--------|-----------|--------|-----------|
| 1024 | 331004 | 169565 | 33.91% | 4.85 | 14 |
| 4096 | 246942 | 255440 | 51.09% | 4.179 | 28 |
| 8192 | 196319 | 308746 | 61.75% | 3.67 | 46 |
| 16384 | 141795 | 369294 | 73.86% | 3.177 | 82 |
| 32768 | 93589 | 432375 | 86.48% | 2.402 | 154 |

-
- 1、观察表可得，当缓冲区越大，磁盘 IO 次数越少、命中数越高，但是文件访问时间会上升，等到缓冲区上升到一定大小时，访问时间又开始减小。说明，为了提升文件访问效率不，能盲目增加缓冲区的大小。
 - 2、随着 `BUFSIZE` 的增大，IO 数量将下降到一个稳定值，命中数、命中率页上升到了一个稳定值，即 `BUFSIZE` 继续增大将不会继改善 IO 性能。
 - 3、所以要合理的设计缓冲区大小。过度增大缓冲区，一方面占用了更多主存储器的宝贵资源；另一方面，在缓冲区置换页面过程中，遍历帧占用时间也会随之增大，甚至会带来负作用。
 - 4、

六、源码

实验源码在压缩包中，也包括了可执行文件。