电子科技大学

作 业 报 告

学生姓名： 冯文森 **学 号：201522010534** **指导教师：李林**

学生 E-mail：1071881690@qq.com

一、作业名称

编写带缓存的文件操作类

二、作业要求

编写一个文件操作的封装类，其要求如下：

需要提供 open/read/write/lseek/close 等函数的封装函数
该类要提供数据缓存服务。
调用该类的写操作接口时，数据要首先写到缓存，然后再根据策略写到文件中。
调用该类的读操作接口时，该类能根据策略缓存读出的数据
读写缓存要保持一致性

我的理解：
根据作业要求，编写封装类则是：

1. 方便用户使用该封装类进行文件操作，原生的 linux 文件操作过于繁杂，编写文件封装类有利于用户使用；
2. 编写文件封装类时可将一些函数封装在构造函数和析构函数中，例如将 close 函数放置到析构函数中，用户不用每次文件读写之后都要手动关闭文件，这是利用 C++的析构函数的优点
3. read/write 函数每次读写时都要陷入内核，编写带缓冲的读写方法更有效率
4. 缓存的大小设计，直接使用教材上的给的缓存大小为 4096 字节的方案；

三、设计与实现

实验步骤：
根据要求，设计一个 FileHandler 类，FileHandler 类主要有以下。
public 函数：

1) FileHandler(char *): 构造函数，接受一个字符串，字符串即为文件名;
2) ～FileHandler(): 析构函数，delete 掉已分配的内存，同时关闭文件描述符；

3) void OpenFile(int mode): 打开文件操作，以 mode 模式打开文件；
4) void ReadFile(char* userBuf, size_t n)：在文件中读取 n 个字符，和 private 中 int ReadFlush()结合使用；
5) void WriteFile(const char* userBuf, size_t n): 向文件中写入 n 个字符，和 private 函数中的 void WriteFlush()函数结合使用实现缓存写；
6) void SeekFile(off_t n, int): 将文件定位到 n 处。

Private 函数：

1) FileHandler():禁用无参构造函数，以下几个同理。
2) FileHandler& operater=(const Handler&):
3) FileHandler(const Handler&).

Private 函数：

1）int ReadFlush(): 实现缓存读的主要函数；
2）void WriteFlush():实现缓存写的主要函数；

private 变量：

1) char *m_FileBuffer: 读或写的缓存区，分配大小为 4096 字节；
2) int m_Fd: 文件描述符；
3) unsigned int m_nUsedBytesForBuffer: 判断缓存大小的变量，在 read 和 write 中有不同含义，后面介绍；
4) unsigned int m_nStartReadPosition: 缓存读的起始位置；
5) char *FileName: 文件名；
6) bool m_nFileOpenForWrite: 判断是否为写文件。

设计细节：

1）构造函数接受文件名后，便为 FileName 分配内存，将文件名拷贝到 FileName，同时为缓存区 m_FileBuffer 分配内存，初始化其它变量。
2）以 mode 模式调用 OpenFile(int)，即以 mode 模式打开文件；
3）如果用户是写文件：
1. 如果写内容大小大于缓存总大小，直接使用 write 函数将内容写入文件；
2. 否则如果写内容大小大于缓存区中剩余大小，则先调用 WriteFlush（）函数将缓存区中内容写入文件，再将要写入的内容拷贝到缓冲区；
3. 否则直接将内容拷贝到缓冲区。
4）如果是读文件：

注意：变量中 m_nStartReadPosition 指向缓冲区中未读取内容的起始位置，而 m_nUsedBytesForBuffer 指向缓存区中未读内容的结束位置。

1. 判断将要读内容的大小，如果内容大小大于缓冲区总大小，直接使用 read 函数读取；
2. 否则如果将要读的内容大小小于缓冲区中未读大小（m_nUsedBytesForBuffer – m_nStartReadPosition），则直接将缓冲区中该部分内容直接拷贝给用户；
3. 否则先将缓冲区中所有未读内容拷贝给用户，再调用 ReadFlush()函数读取 4096 字节内容到缓冲区，再从缓冲区中拷贝部分内容到用户内存中剩余部分。

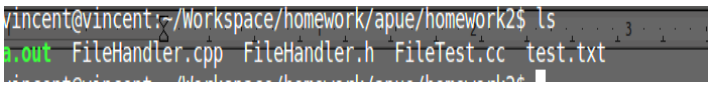
5）FileSeek()函数主要包装了下 lseek 函数，简单处理错误部分；
6）虚构函数正如函数介绍部分。

四、测试

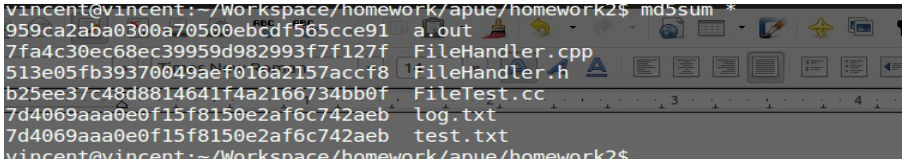
测试结果图：

测试文件读写完整性：

1. 将 man man 写入 test.txt 文件中。



2. 编写测试程序，读取 test.txt 的内容写入 log.txt 文件中，使用 md5sum 查看两个文件的校验码。

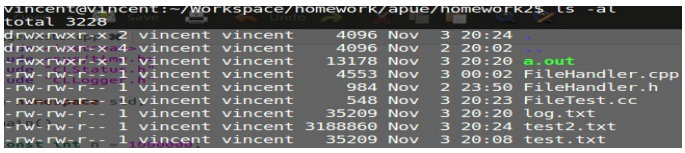


从上图中可看出 test.txt 和 log.txt 文件校验码相同，即文件读写完整。

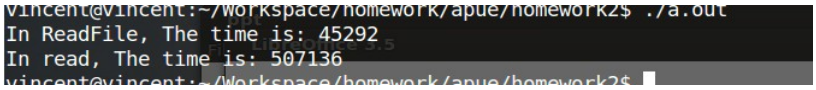
测试文件读效率：

方案：分别使用本包装类中的 ReadFile 函数和原生 read 函数读取 1000000 个字符，每次读取一个字符，测试两次读的时间。

1. 创建测试文件 test2.txt，将 man man 反复写入文件中，得到如下图的 test2.txt 文件。



2. 编写测试程序，测试两次时间：

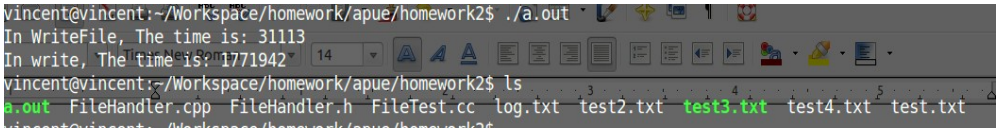


从上图看出 ReadFile 函数比 read 快一个量级。

测试写文件效率：

和测试读文件同样方案，使用两种方法将 1000000 个字符写入文件中，测试两种方法的时间。

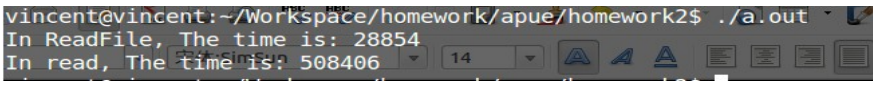
从上图中可看出，WriteFile 函数比 write 函数快 2 个量级。



分析：

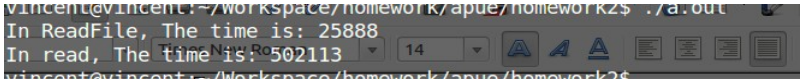
ReadFile 函数仅比 read 快一个量级，可能是和缓存大小有关系，故我将改变缓存大小后再次测试，

1. 缓存大小改为 2*4096 字节。



果然，速度比原来 4096 字节快了一倍。

2. 缓存大小改为 3*4096 比特：



基本上和 2*4096 一致。

后面的如 4*4096 经过测试，基本和 2*4096 一致。

五、对本课程或本作业的建议和意见

六、附录

FileHandler.h:

```
#ifndef _FILEHANDLER_H
#define _FILEHANDLER_H
#include<cstring>
#include<errno.h>
#include<unistd.h>
#include<fcntl.h>
const int BUFFER_SIZE = 4096;
const int FILE_NAME_LENGTH = 128;

class FileHandler {
public:
    explicit FileHandler(char *); //构造函数
    ~FileHandler(); 析构函数
    void OpenFile(int); //打开文件
    void ReadFile(char*, size_t); //读取文件
    void WriteFile(const char*, size_t); //写文件
    void SeekFile(off_t, int); //定位文件
private:
    FileHandler(const FileHandler&);    //
    FileHandler& operator=(const FileHandler&);
    FileHandler();
private:
    int ReadFlush();    //read flush 读缓存实现
    void WriteFlush();    //write flush 写缓存实现
    char *m_FileBuffer;    //read or write buffer 缓存
```

```
    int m_Fd;    //file handler 文件描述符
    //unsigned long m_nSeekPosition;
    unsigned int m_nUsedBytesForBuffer;  //缓存位置
    unsigned int m_nStartReadPositionForBuffer; //读缓存中，未读缓存起始位置
    char *m_FileName;  //  read or write file name  文件名
   //是否打开文件未写，在析构函数中再次调用将缓存中剩余内容写入文件
  bool m_nFileOpenForWrite;
    //bool m_nFileOpenForRead;
};
#endif
```

FileHandler.cpp:

```
/*  Author: vincent
 *  File name: FileHandler.cpp
 *  description: a wrap class for file handler
 */
#include<iostream>
#include"FileHandler.h"
using namespace std;

//构造函数
FileHandler::FileHandler(char *fileName) {
    m_FileBuffer = new char[BUFFER_SIZE];       //allocate BUFFER_SIZE bytes for buffer
    m_FileName = new char[FILE_NAME_LENGTH];     //allocate FILE_NAME_LENGTH bytes for
file name
    memset(m_FileName, 0, sizeof(m_FileName));
    memset(m_FileBuffer, 0, sizeof(m_FileBuffer));
    memcpy(m_FileName, fileName, strlen(fileName));   //copy the file name to m_FileName
    m_nUsedBytesForBuffer = 0;
    m_nStartReadPositionForBuffer = 0;
    m_nFileOpenForWrite = false;
    //m_nFileOpenForRead = false;
}

void FileHandler::OpenFile(int mode) {
    //open file with mode mode, and get the file descriptor
    m_Fd = open(m_FileName, mode);
    if (m_Fd == -1) {
        cout << "Open file error: " << strerror(errno) << endl;
        return;
    }
}

void FileHandler::WriteFile(const char *userBuf, size_t nBytes) {
```

```
        //
    if (!m_nFileOpenForWrite)
        m_nFileOpenForWrite = true;
    if (userBuf == 0)
        return;
    if (m_FileBuffer == 0)
        return;
    unsigned int nLeftRoom = BUFFER_SIZE – m_nUsedBytesForBuffer;     //缓存中剩余大小
    unsigned int lenMsg = strlen(userBuf);

    //如果写入大小大于BUFFER_SIZE，则直接写入文件
    if (lenMsg > BUFFER_SIZE) {
        int ret;
        ret = write(m_Fd, userBuf, lenMsg);
        if (ret == -1) {
            cout << "Write error: " << strerror(errno) << endl;
            return;
        }
    }

    //if m_FileBuffer can not load more message, then Flush, that is to say write buffer to file.
    if (lenMsg > nLeftRoom) {
        WriteFlush();
    }

    //copy the message to the m_FileBuffer
    memcpy(m_FileBuffer + m_nUsedBytesForBuffer, userBuf, lenMsg);
    m_nUsedBytesForBuffer += lenMsg;   //add the m_nUsedBytesForBuffer
}

void FileHandler::ReadFile(char* userBuf, size_t nBytes) {
  //  if (!m_nFileOpenForRead)
  //     m_nFileOpenForRead = true;
    if (userBuf == 0)
        return;
    if (nBytes == 0)
        return;

    // if read bytes larger thar BUFFER_SIZE, read directly
    if (nBytes > BUFFER_SIZE) {
        int ret = read(m_Fd, userBuf, sizeof(userBuf));
        if (ret == -1) {
            cout << "Read error: " << strerror(errno) << endl;
            return;
```

```
        }
    }

//如果读取的大小小于缓存中未读大小，直接拷贝到用户缓存中
    if (nBytes <= (m_nUsedBytesForBuffer - m_nStartReadPositionForBuffer)) {
        memcpy(userBuf, m_FileBuffer + m_nStartReadPositionForBuffer, nBytes);
        m_nStartReadPositionForBuffer += nBytes;
    } else {   //否则， 调用 ReadFlush
        int len = m_nUsedBytesForBuffer - m_nStartReadPositionForBuffer;
        memcpy(userBuf, m_FileBuffer + m_nStartReadPositionForBuffer, len);
        m_nStartReadPositionForBuffer = 0;
        int ret = ReadFlush();
        if (ret == 0) {   //如果文件读取完毕，则提示并返回
            cout << "Read over!" << endl;
            return;
        }

        int cpyLen = nBytes - len;
        if (ret < cpyLen)
            cpyLen = ret;
        memcpy(userBuf + len, m_FileBuffer, cpyLen);
        m_nStartReadPositionForBuffer += cpyLen;
    }
}

//seek file
void FileHandler::SeekFile(off_t offSet, int whence) {

    int ret;
    ret = lseek(m_Fd, offSet, whence);
    if (ret == -1) {
        cout << "Seek file error: " << strerror(errno) << endl;
        return;
    }
}

//写缓存
void FileHandler::WriteFlush(){

    if (m_Fd == -1)
        return;
    if (m_nUsedBytesForBuffer == 0)
        return;
    if (m_FileBuffer == 0)
```

```
        return;
        //将缓存中内容写入文件
    int ret = write(m_Fd, m_FileBuffer, m_nUsedBytesForBuffer);
    if (ret == -1) {
        cout << "Write file error: " << strerror(errno) << endl;
        return;
    }
    m_nUsedBytesForBuffer = 0;
}

//读缓存实现
int FileHandler::ReadFlush() {

    int ret;
//一次读取 BUFFER_SIZE 大小
    ret = read(m_Fd, m_FileBuffer, BUFFER_SIZE);
    if (ret == -1) {
        cout << "Read error: " << strerror(errno) << endl;
        return -1;
    }

    //EOF   文件末尾
    if (ret == 0) {
        m_nUsedBytesForBuffer = 0;
        m_nStartReadPositionForBuffer = 0;
        return 0;
    }

    m_nUsedBytesForBuffer = ret;
    m_nStartReadPositionForBuffer = 0;
    return ret;
}

//清理战场
FileHandler::~FileHandler() {
    int ret;
//判断如果是写文件，则将缓存中剩余部分写入文件
    if (m_nFileOpenForWrite)
        WriteFlush();

    delete [] m_FileBuffer;  //deallocate
    delete [] m_FileName;    //deallocate
    ret = close(m_Fd);  //close file descriptor
    if (ret == -1) {
```

```
        cout << "Close file error: " << strerror(errno) << endl;
    }
}
```

测试文件，FileTest.cpp：

```
#include"FileHandler.h"
#include<iostream>
#include<sys/time.h>

const int MAX_CHAR = 1000000;
using namespace std;

int main(){

 // 测试读写完整性.
 /*
   FileHandler file("test.txt"), file2("log.txt");
   char buf[1024];
   memset(buf, 0, sizeof(buf));
   file.OpenFile(O_RDWR);
   file2.OpenFile(O_RDWR);
   for (int i = 0; i < 100; i++) {
      file.ReadFile(buf, 1024);
      file2.WriteFile(buf, 1024);
      //cout << buf ;
      memset(buf, 0, sizeof(buf));
   }
*/


//测试读文件效率**********

   FileHandler *file = new FileHandler("test2.txt");
   file->OpenFile(O_RDONLY);
   char buf[2];
   memset(buf, 0, sizeof(buf));
   struct timeval tv1, tv2, tv3, tv4;
   gettimeofday(&tv1, 0);
   for (int i = 0; i < MAX_CHAR; ++i)
      file->ReadFile(buf, 1);
   gettimeofday(&tv2, 0);
   long t = 1000000*(tv2.tv_sec - tv1.tv_sec) + (tv2.tv_usec-tv1.tv_usec);
```

```
    cout << "In ReadFile, The time is: " << t << endl;
    delete file;

    memset(buf, 0, sizeof(buf));
    int fd = open("test2.txt", O_RDONLY);
    gettimeofday(&tv3, 0);
    for (int i = 0; i < MAX_CHAR; ++i)
      read(fd, buf, 1);
    gettimeofday(&tv4, 0);
    t = 1000000*(tv4.tv_sec - tv3.tv_sec) + (tv4.tv_usec-tv3.tv_usec);
    cout << "In read, The time is: " << t << endl;
    close(fd);

    //*****************************



/*

    //测试写文件效率

    FileHandler *file = new FileHandler("test3.txt");
    file->OpenFile(O_WRONLY|O_CREAT);
    char buf[2];
    memset(buf, 0, sizeof(buf));
    struct timeval tv1, tv2, tv3, tv4;
    gettimeofday(&tv1, 0);
    for (int i = 0; i < MAX_CHAR; ++i)
       file->WriteFile("1", 1);
    gettimeofday(&tv2, 0);
    long t = 1000000*(tv2.tv_sec - tv1.tv_sec) + (tv2.tv_usec-tv1.tv_usec);
    cout << "In WriteFile, The time is: " << t << endl;
    delete file;

    memset(buf, 0, sizeof(buf));
    int fd = open("test4.txt", O_WRONLY | O_CREAT);
    gettimeofday(&tv3, 0);
    for (int i = 0; i < MAX_CHAR; ++i)
      write(fd, "2", 1);
    gettimeofday(&tv4, 0);
    t = 1000000*(tv4.tv_sec - tv3.tv_sec) + (tv4.tv_usec-tv3.tv_usec);
    cout << "In write, The time is: " << t << endl;
    close(fd);
    //************************************************************
```

```
*/
}
```