**电 子 科 技 大 学**

作 业 报 告

# 学生姓名：王冉恒 学 号：202422900232 指导教师：李林学生E-mail：[1976569841@qq.com](mailto:djhuang_1@qq.com)

**一、作业名称**

编写一个带缓存的文件操作类

# 二、作业要求

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。

# 三、设计与实现

本作业文件类的定义如下：

class CachedFile {public:

CachedFile(size\_t buffer\_size = 8192) : buffer\_size\_(buffer\_size), pos\_(0) {

buffer\_.resize(buffer\_size\_);

}

~CachedFile() {

close();

}

bool open(const std::string &filename, const std::string &mode) {

mode\_ = mode;

file\_.open(filename, std::fstream::binary | std::fstream::out | std::fstream::in | std::fstream::app);

if (!file\_.is\_open()) {

std::cerr << "Failed to open file: " << filename << std::endl;

return false;

}

return true;

}

size\_t read(char \*ptr, size\_t size) {

size\_t total\_read = 0;

while (total\_read < size) {

if (pos\_ == buffer\_size\_) {

file\_.read(buffer\_.data(), buffer\_size\_);

pos\_ = 0;

if (file\_.gcount() == 0) break; *// End of file or error*

}

size\_t to\_copy = std::min(size - total\_read, buffer\_size\_ - pos\_);

std::memcpy(ptr + total\_read, buffer\_.data() + pos\_, to\_copy);

total\_read += to\_copy;

pos\_ += to\_copy;

}

return total\_read;

}

size\_t write(const char \*ptr, size\_t size) {

size\_t total\_written = 0;

while (total\_written < size) {

size\_t to\_copy = std::min(size - total\_written, buffer\_size\_ - pos\_);

std::memcpy(buffer\_.data() + pos\_, ptr + total\_written, to\_copy);

total\_written += to\_copy;

pos\_ += to\_copy;

if (pos\_ == buffer\_size\_) {

file\_.write(buffer\_.data(), buffer\_size\_);

pos\_ = 0;

}

}

return total\_written;

}

bool lseek(long offset, int whence) {

if (whence == SEEK\_SET) {

file\_.seekg(offset, std::ios::beg);

file\_.seekp(offset, std::ios::beg);

} else if (whence == SEEK\_CUR) {

file\_.seekg(offset, std::ios::cur);

file\_.seekp(offset, std::ios::cur);

} else if (whence == SEEK\_END) {

file\_.seekg(offset, std::ios::end);

file\_.seekp(offset, std::ios::end);

}

pos\_ = 0;

return file\_.good();

}

void close() {

if (file\_.is\_open()) {

if (pos\_ > 0) {

file\_.write(buffer\_.data(), pos\_);

}

file\_.close();

}

}

private:

std::fstream file\_;

std::vector<char> buffer\_;

size\_t buffer\_size\_;

size\_t pos\_;

std::string mode\_;

};

其中实现的 open、write、lseek、read、close 函数的功能对应 glibc 中 open、write、

lseek、read、close 函数，并且实现了文件缓存。该类维护了一个vector类型的buffer来实现缓存功能，当像文件写入数据时，会先写入这个buffer中，当buffer写满后，再向文件写入。当要关闭文件和要读取文件时，就会检查buffer是否还存在数据，如果还存在，就先将buffer中中的数据写入文件，再进行关闭或读取。

具体实现：

Open函数：输入文件名和打开模式之后，调用fstream的open函数，对该文件进行打开，打开成功返回true，失败返回false。

Read函数：调用函数时要输入读取文件存储位置和读取数据的数量，定义一个变量total\_read来记录总共读取了多少数据，当已读数据少于要读的数据时，就循环读，每次读取数据时，先存到buffer中，再复制到存储数据的变量里。

Write函数：调用函数时要指定写入文件的数据存储位置和写入数据的数量，定义一个变量total\_writeten来记录总共写了多少数据，当已写数据少于要写的数据时，就循环写，每次写数据时，先存到buffer中，buffer写满后，再写到文件里。

Lseek函数：通过指定的文件位置和偏移量，在函数中调用glibc的seekp和seekg函数，来改变文件读写指针的位置。

Close函数：调用该函数时，文件类会先将存在buffer中还没来得及写入文件的数据写入文件，之后调用glibc的close函数来关闭文件。

# 四、测试

main函数内容如下：

int main() {

CachedFile cached\_file;

if (cached\_file.open("data.txt", "w+")) {

cached\_file.write("Hello, World!", 13);

cached\_file.lseek(0, SEEK\_SET);

char buffer[14];

cached\_file.read(buffer, 13);

buffer[13] = '\0';

std::cout << "Read from file: " << buffer << std::endl;

cached\_file.close();

} else {

std::cerr << "Failed to open file." << std::endl;

}

return 0;

}

Main函数定义了一个刚刚实现的带缓存的文件操作类，接下来依次执行打开data.txt文件、向这个文件写入“Hello，world”字符串、移动文件读写指针到文件开头、在文件中读取13个字节的内容、关闭文件。

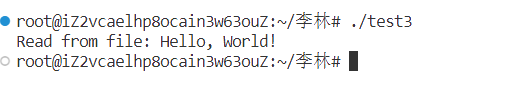
编译：

g++ -o cache\_file cache\_file.cpp

执行：

./cache\_file

显示：



如图所示，成功写入后读取data.txt 文件内容

open、write、read、lseek、close 功能均正常。

# 五、附录

#include <iostream>

#include <fstream>

#include <vector>

#include <cstring>

class CachedFile {public:

CachedFile(size\_t buffer\_size = 8192) : buffer\_size\_(buffer\_size), pos\_(0) {

buffer\_.resize(buffer\_size\_);

}

~CachedFile() {

close();

}

bool open(const std::string &filename, const std::string &mode) {

mode\_ = mode;

file\_.open(filename, std::fstream::binary | std::fstream::out | std::fstream::in | std::fstream::app);

if (!file\_.is\_open()) {

std::cerr << "Failed to open file: " << filename << std::endl;

return false;

}

return true;

}

size\_t read(char \*ptr, size\_t size) {

size\_t total\_read = 0;

while (total\_read < size) {

if (pos\_ == buffer\_size\_) {

file\_.read(buffer\_.data(), buffer\_size\_);

pos\_ = 0;

if (file\_.gcount() == 0) break; *// End of file or error*

}

size\_t to\_copy = std::min(size - total\_read, buffer\_size\_ - pos\_);

std::memcpy(ptr + total\_read, buffer\_.data() + pos\_, to\_copy);

total\_read += to\_copy;

pos\_ += to\_copy;

}

return total\_read;

}

size\_t write(const char \*ptr, size\_t size) {

size\_t total\_written = 0;

while (total\_written < size) {

size\_t to\_copy = std::min(size - total\_written, buffer\_size\_ - pos\_);

std::memcpy(buffer\_.data() + pos\_, ptr + total\_written, to\_copy);

total\_written += to\_copy;

pos\_ += to\_copy;

if (pos\_ == buffer\_size\_) {

file\_.write(buffer\_.data(), buffer\_size\_);

pos\_ = 0;

}

}

return total\_written;

}

bool lseek(long offset, int whence) {

if (whence == SEEK\_SET) {

file\_.seekg(offset, std::ios::beg);

file\_.seekp(offset, std::ios::beg);

} else if (whence == SEEK\_CUR) {

file\_.seekg(offset, std::ios::cur);

file\_.seekp(offset, std::ios::cur);

} else if (whence == SEEK\_END) {

file\_.seekg(offset, std::ios::end);

file\_.seekp(offset, std::ios::end);

}

pos\_ = 0;

return file\_.good();

}

void close() {

if (file\_.is\_open()) {

if (pos\_ > 0) {

file\_.write(buffer\_.data(), pos\_);

}

file\_.close();

}

}

private:

std::fstream file\_;

std::vector<char> buffer\_;

size\_t buffer\_size\_;

size\_t pos\_;

std::string mode\_;

};

int main() {

CachedFile cached\_file;

if (cached\_file.open("data.txt", "w+")) {

cached\_file.write("Hello, World!", 13);

cached\_file.lseek(0, SEEK\_SET);

char buffer[14];

cached\_file.read(buffer, 13);

buffer[13] = '\0';

std::cout << "Read from file: " << buffer << std::endl;

cached\_file.close();

} else {

std::cerr << "Failed to open file." << std::endl;

}

return 0;

}