Windows操作系统

C/C++ 程序实验

首都师范大学 信息工程学院

系统分析与管理实验室

## 实验九 Windows 快速文件系统

### 一、背景知识

  众所周知，CPU是整个计算机系统中运算速度最快的部分，而外部设备是最慢的部分，它们之间存在着很大的差别。然而，CPU却时时刻刻可能要求访问外设。如果CPU的每次操作都必须等待外设完成，那么CPU宝贵的运行时间就会大大浪费。随着现代计算机技术的发展，大多数现代操作系统都对这个问题进行了处理：下面就介绍两种Windows 中解决这个不匹配问题的方法：高速缓存和异步传输。

### 二、实验目的

1、文件高速缓存

文件高速理存是CPU访问外设的一个“中间设备”。说是设备，其实它不是真正的物理“设备”，而是一种核心级内存映像机制。由于它被设置在内存中，因此速度非常快。可以部分解决CPU与硬盘速度差异的问题。文件系统的驱动程序通过调用“高速缓存管理程序”来使用文件高速缓存，然后高速缓存管理程序执行高速缓存的处理工作。

文件高速缓存的原理是：假设一个进程读了文件的第一个字节，它常常会按照顾序读第二个、第三个字节，一直到读出所有的字节。利用这个原理可以进行“预取”，也就是说．在进程没请求读磁盘之前就先把文件读出来并放到高速缓存中。这样，当进程请求访问磁盘时，高速缓存可以快速地把已经取到内存中的文件内容直接送给进程使用，从而，大大加速了访问磁盘的速度。另外，由于一个文件可能会被多次读入。因此可以在第一次读入后，将文件数据保存在高速缓存中。这样，下次再读时就不必再从硬盘而可以从缓存中读取。利用LRU(Least Recently Used，最近最少使用)的原则，可以将不常使用的文件从缓存中删除以节省高速缓存空间。

另外，文件高速缓存还有一个“事后写”的机制。具体地讲，如果一个进程要求写磁盘，它首先把要写的内容交给高速缓存。而高速缓存并不马上把它写到磁盘上，而是寻找CPU空闲的时间来进行写操作。这样，要写磁盘的进程就可以不必等待磁盘写完毕以后再继续工作。这也就节省了整个进程的执行时间。这里需要说明的是，如果有另外一个进程要访问还没有被写入磁盘的文件时．高速缓存管理程序可以使这个进程直接读高速缓存里面新的即将要写入的文件内容，而不是磁盘上的旧内容，从而保证了文件内容的一致性。

2、异步传输

与文件高速缓存不同，文件的异步传输是一种改变指令执行顺序的机制。在以往的操作系统中，指令都是顺序执行的，下一条指令必须在上一条指令执行完毕后才能执行。因此，如果CPU遇到一条访盘指令，那么它就必须等待缓慢的磁盘访问结束以后才能进行后续的工作。如果它后面的指令并不依赖于访盘操作时，这个等待就显得很没有必要。Windows 中使用了一种异步传输的机制来解决这个问题。它通过设置打开文件时的一个标志位来使进程不等待读写文件操作而继续执行。当后续指令必须用到磁盘访问的结果数据时，它再通过一条wait指令进行等待。这样，在访盘指令和等待指令之间的指令就可以与磁盘访问同时进行了，从而大大加快了系统的整体速度。

### 三、工具/准备工作

在开始本实验之前，请回顾教科书的相关内容。

需要做以下准备：

1) 一台运行Windows 操作系统的计算机。

2) 计算机中需安装Microsoft Visual Studio Code。

### 四、实验内容与步骤

#### 1. 快速文件系统

**步骤1**：登录进入Windows 。

**步骤2**：在“开始”菜单中单击“程序”-“Microsoft Visual Studio Code”，进入Visual C++窗口。

**步骤3：**新建项目名为“9-1”，并且新建项“9-1.cpp”。

**清单9-1 快速文件系统**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*-------------------------------------------------------------

/\*三种模式

/\* 1. FILE\_FLAG\_NOBUFFER

/\* 2. FILE\_FLAG\_SEQUENTIAL\_SCAN

/\* 3. FILE\_FLAG\_BUFFERING|FILE\_FLAG\_OVERLAPPED

/\*------------------------------------------------------------

/\*五种操作

/\* 1. charactor +1

/\* 2. charactor -1

/\* 3. charactor -32

/\* 4. charactor +32

/\* 5. charactor \*1

/\*---------------------------------------------------------------

\*/

#include<iostream>

#include<windows.h>

using namespace std;

//三种模式

void filter\_nobuffer(char\* source, char\* sink, void(\*func)(char\* addr));

void filter\_sequen(char\* source, char\* sink, void(\*func)(char\* addr));

void filter\_overlp(char\* source, char\* sink, void(\*func)(char\* addr));

//五种不同功能的操作

void f1(char\* addr);

void f2(char\* addr);

void f3(char\* addr);

void f4(char\* addr);

void f5(char\* addr);

#define BUFFER\_SIZE 1024 //定义缓冲区的大小，这里设为1024字节

char\* buffer; //这里的缓冲区被定义成char型

void main()

{

//分配缓冲区

buffer = new char[BUFFER\_SIZE];

//记录执行filter函数的开始时间

DWORD tick;

//用于三种模式各自的平均时间

DWORD nobuffer\_average\_time = 0;

DWORD sequen\_average\_time = 0;

DWORD overlp\_average\_time = 0;

//采用无缓存模式调用filter函数10次

cout << "\*无文件高速缓存模式正在运行......" << endl;

DWORD nobuffer\_start\_time = GetTickCount();

tick = nobuffer\_start\_time;

filter\_nobuffer("source.txt", "nobuffer\_1.txt", f1);

cout << "nobuffer 0-1:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_nobuffer("nobuffer\_1.txt", "nobuffer\_2.txt", f2);

cout << "nobuffer 1-2:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_nobuffer("nobuffer\_2.txt", "nobuffer\_3.txt", f3);

cout << "nobuffer 2-3:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_nobuffer("nobuffer\_3.txt", "nobuffer\_4.txt", f4);

cout << "nobuffer 3-4:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_nobuffer("nobuffer\_4.txt", "nobuffer\_5.txt", f5);

cout << "nobuffer 4-5:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_nobuffer("nobuffer\_5.txt", "nobuffer\_6.txt", f1);

cout << "nobuffer 5-6:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_nobuffer("nobuffer\_6.txt", "nobuffer\_7.txt", f2);

cout << "nobuffer 6-7:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_nobuffer("nobuffer\_7.txt", "nobuffer\_8.txt", f3);

cout << "nobuffer 7-8:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_nobuffer("nobuffer\_8.txt", "nobuffer\_9.txt", f4);

cout << "nobuffer 8-9:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_nobuffer("nobuffer\_9.txt", "nobuffer\_10.txt", f5);

DWORD nobuffer\_end\_time = GetTickCount();

cout << "nobuffer 9-10:" << nobuffer\_end\_time - tick << " ms." << endl;

//采用高速缓存模式调用filter函数10次

cout << "\*使用文件高速缓存模式正在运行......" << endl;

DWORD sequen\_start\_time = GetTickCount();

tick = sequen\_start\_time;

filter\_sequen("source.txt", "sequen\_1.txt", f1);

cout << "sequen 0-1:" << GetTickCount() - tick << " ms. " << endl;

tick = GetTickCount();

filter\_sequen("sequen\_1.txt", "sequen\_2.txt", f2);

cout << "sequen 1-2:" << GetTickCount() - tick << " ms. " << endl;

tick = GetTickCount();

filter\_sequen("sequen\_2.txt", "sequen\_3.txt", f3);

cout << "sequen 2-3:" << GetTickCount() - tick << " ms. " << endl;

tick = GetTickCount();

filter\_sequen("sequen\_3.txt", "sequen\_4.txt", f4);

cout << "sequen 3-4:" << GetTickCount() - tick << " ms. " << endl;

tick = GetTickCount();

filter\_sequen("sequen\_4.txt", "sequen\_5.txt", f5);

cout << "sequen 4-5:" << GetTickCount() - tick << " ms. " << endl;

tick = GetTickCount();

filter\_sequen("sequen\_5.txt", "sequen\_6.txt", f1);

cout << "sequen 5-6:" << GetTickCount() - tick << " ms. " << endl;

tick = GetTickCount();

filter\_sequen("sequen\_6.txt", "sequen\_7.txt", f2);

cout << "sequen 6-7:" << GetTickCount() - tick << " ms. " << endl;

tick = GetTickCount();

filter\_sequen("sequen\_7.txt", "sequen\_8.txt", f3);

cout << "sequen 7-8:" << GetTickCount() - tick << " ms. " << endl;

tick = GetTickCount();

filter\_sequen("sequen\_8.txt", "sequen\_9.txt", f4);

cout << "sequen 8-9:" << GetTickCount() - tick << " ms. " << endl;

tick = GetTickCount();

filter\_sequen("sequen\_9.txt", "sequen\_10.txt", f5);

DWORD sequen\_end\_time = GetTickCount();

cout << "sequen 9-10:" << sequen\_end\_time - tick << " ms. " << endl;

//采用异步模式调用filter函数10次

cout << "异步传输模式正在运行......" << endl;

DWORD overlp\_start\_time = GetTickCount();

tick = overlp\_start\_time;

filter\_overlp("source.txt", "overlp\_1.txt", f1);

cout << "overlp 0-1:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_overlp("overlp\_1.txt", "overlp\_2.txt", f2);

cout << "overlp 1-2:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_overlp("overlp\_2.txt", "overlp\_3.txt", f3);

cout << "overlp 2-3:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_overlp("overlp\_3.txt", "overlp\_4.txt", f4);

cout << "overlp 3-4:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_overlp("overlp\_4.txt", "overlp\_5.txt", f5);

cout << "overlp 4-5:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_overlp("overlp\_5.txt", "overlp\_6.txt", f1);

cout << "overlp 5-6:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_overlp("overlp\_6.txt", "overlp\_7.txt", f2);

cout << "overlp 6-7:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_overlp("overlp\_7.txt", "overlp\_8.txt", f3);

cout << "overlp 7-8:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_overlp("overlp\_8.txt", "overlp\_9.txt", f4);

cout << "overlp 8-9:" << GetTickCount() - tick << " ms." << endl;

tick = GetTickCount();

filter\_overlp("overlp\_9.txt", "overlp\_10.txt", f5);

DWORD overlp\_end\_time = GetTickCount();

cout << "overlp 9-10:" << overlp\_end\_time - tick << " ms." << endl;

//输出三种模式下的平均时间以做对比

cout << "\*三种模式的平均用时如下：" << endl;

cout << ".无模式高速缓存模式平均用时：" << (nobuffer\_end\_time - nobuffer\_start\_time) / 10 << " ms." << endl;

cout << ".使用文件高速缓存模式平均用时：" << (sequen\_end\_time - sequen\_start\_time) / 10 << " ms." << endl;

cout << ".异步传输模式平均用时：" << (overlp\_end\_time - overlp\_start\_time) / 10 << " ms." << endl << endl;

system("pause");

return;

}

/\*------------------------------------------------------------------\*/

//对文件内容进行的5种操作

//f1 +1

//f2 -1

//f3 \*1

//f4 >>

//f5 <<

void f1(char\* addr) { \*addr = (unsigned char)\*addr + 1; }

void f2(char\* addr) { \*addr = (unsigned char)\*addr - 1; }

void f3(char\* addr) { \*addr = (unsigned char)\*addr \* 1; }

void f4(char\* addr) { \*addr = (unsigned char)\*addr >> 1; }

void f5(char\* addr) { \*addr = (unsigned char)\*addr << 1; }

/\*-------------------------------------------------------------\*/

//没有文件高速缓存的filter函数

void filter\_nobuffer(char\* source, char\* sink, void(\*func)(char\* addr))

{

HANDLE handle\_src, handle\_dst; //定义原文件与目标文件的句柄

BOOL cycle; //用来判断一个缓冲区是否被写满

DWORD NumberOfBytesRead, NumberOfBytesWrite, index; //读的字节数、写的字节数

//打开原文件

handle\_src = CreateFile(source, GENERIC\_READ, NULL, NULL, OPEN\_EXISTING, FILE\_FLAG\_NO\_BUFFERING, NULL);

//创建目标文件

handle\_dst = CreateFile(sink, GENERIC\_WRITE, NULL, NULL, CREATE\_ALWAYS, NULL, NULL);

if(handle\_src == INVALID\_HANDLE\_VALUE || handle\_dst == INVALID\_HANDLE\_VALUE)

{

cout << "CreatFile Invocation Error!" << endl;

exit(1);

}

cycle = TRUE;

//用来cycle判断文件什么时候读完

while (cycle)

{

//从原文件读数据送入缓冲区

if (ReadFile(handle\_src, buffer, BUFFER\_SIZE, &NumberOfBytesRead, NULL) == FALSE)

{

cout << "ReadFile Error!" << endl;

exit(1);

}

//当读不满一个缓冲区时，说明达到文件末尾，结束循环

if (NumberOfBytesRead < BUFFER\_SIZE)

cycle = FALSE;

//对文件内容进行的操作

for (index = 0; index < NumberOfBytesRead; index++)

func(&buffer[index]);

//将缓冲区中的数据写入目标文件

if (WriteFile(handle\_dst, buffer, NumberOfBytesRead, &NumberOfBytesWrite, NULL) == FALSE)

{

cout << "WriteFile Error!" << endl;

exit(1);

}

}

//关闭文件句柄

CloseHandle(handle\_src);

CloseHandle(handle\_dst);

}

/\*-------------------------------------------------------------\*/

void filter\_sequen(char\* source, char\* sink, void(\*func)(char\* addr))

{

HANDLE handle\_src, handle\_dst; //定义原文件与目标文件的句柄

BOOL cycle; //用来判断一个缓冲区是否被写满

DWORD NumberOfBytesRead, NumberOfBytesWrite, index; //读的字节数、写的字节数

//打开原文件

handle\_src = CreateFile(source, GENERIC\_READ, NULL, NULL, OPEN\_EXISTING, FILE\_FLAG\_SEQUENTIAL\_SCAN, NULL);

//创建目标文件

handle\_dst = CreateFile(sink, GENERIC\_WRITE, NULL, NULL, CREATE\_ALWAYS, FILE\_FLAG\_SEQUENTIAL\_SCAN, NULL);

if(handle\_src == INVALID\_HANDLE\_VALUE || handle\_dst == INVALID\_HANDLE\_VALUE)

{

cout << "CreatFile Invocation Error!" << endl;

exit(1);

}

cycle = TRUE;

//用来cycle判断文件什么时候读完

while (cycle)

{

//从原文件读数据送入缓冲区

if (ReadFile(handle\_src, buffer, BUFFER\_SIZE, &NumberOfBytesRead, NULL) == FALSE)

{

cout << "ReadFile Error!" << endl;

exit(1);

}

//当读不满一个缓冲区时，说明达到文件末尾，结束循环

if (NumberOfBytesRead < BUFFER\_SIZE)

cycle = FALSE;

//对文件内容进行的操作

for (index = 0; index < NumberOfBytesRead; index++)

func(&buffer[index]);

//将缓冲区中的数据写入目标文件

if (WriteFile(handle\_dst, buffer, NumberOfBytesRead, &NumberOfBytesWrite, NULL) == FALSE)

{

cout << "WriteFile Error!" << endl;

exit(1);

}

}

//关闭文件句柄

CloseHandle(handle\_src);

CloseHandle(handle\_dst);

}

/\*-------------------------------------------------------------\*/

void filter\_overlp(char\* source, char\* sink, void(\*func)(char\* addr))

{

HANDLE handle\_src, handle\_dst; //定义原文件与目标文件的句柄

BOOL cycle; //用来判断一个缓冲区是否被写满

DWORD NumberOfBytesRead, NumberOfBytesWrite, index, dwError; //读的字节数、写的字节数

OVERLAPPED overlapped; //overlapped 结构

//打开原文件

handle\_src = CreateFile(source, GENERIC\_READ, NULL, NULL, OPEN\_EXISTING, FILE\_FLAG\_NO\_BUFFERING | FILE\_FLAG\_OVERLAPPED, NULL);

//创建目标文件

handle\_dst = CreateFile(sink, GENERIC\_WRITE, NULL, NULL, CREATE\_ALWAYS, NULL, NULL);

if(handle\_src == INVALID\_HANDLE\_VALUE || handle\_dst == INVALID\_HANDLE\_VALUE)

{

cout << "CreatFile Invocation Error!" << endl;

exit(1);

}

cycle = TRUE;

//对overlapped结构初始化

overlapped.hEvent = NULL;

overlapped.Offset = -BUFFER\_SIZE;

overlapped.OffsetHigh = 0;

//用来cycle判断文件什么时候读完

while (cycle)

{

//计算文件的偏移量

overlapped.Offset = overlapped.Offset + BUFFER\_SIZE;

//从原文件读数据送入缓冲区

if (ReadFile(handle\_src, buffer, BUFFER\_SIZE, &NumberOfBytesRead, &overlapped) == FALSE)

{

switch (dwError = GetLastError())

{

//读文件结尾

case ERROR\_HANDLE\_EOF:

cycle = FALSE;

break;

case ERROR\_IO\_PENDING:

if (GetOverlappedResult(handle\_src, &overlapped, &NumberOfBytesRead, TRUE) == FALSE)

{

cout << "GetOverlappedResult Error!" << endl;

exit(1);

}

break;

default:

break;

}

}

//当不满一个个缓存区时，说明达到文件末尾，结束循环

if (NumberOfBytesRead < BUFFER\_SIZE)

cycle = FALSE;

//对文件内容进行的操作

for (index = 0; index < NumberOfBytesRead; index++)

func(&buffer[index]);

//将缓冲区中的数据写入目标文件

if (WriteFile(handle\_dst, buffer, NumberOfBytesRead, &NumberOfBytesWrite, NULL) == FALSE)

{

cout << "WriteFile Error!" << endl;

exit(1);

}

}

//关闭文件句柄

CloseHandle(handle\_src);

CloseHandle(handle\_dst);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*THE END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**步骤4：**将项目所需txt文件复制到项目文件夹下。

**步骤5：**按“F5”开始调试，注意路径里不要含有中文。

**步骤6：**按暂停按钮可暂停程序的执行，按终止按钮可终止程序的执行。



操作能否正常进行？如果不行，则可能的原因是什么？

操作能够正常进行，如果不能正常进行，可能的原因是因为文件路径中包含中文，或者代码中的字符集有错误。

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

运行结果是：

文本

描述已自动生成

图片 1 运行结果截图

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

结果分析：

使用文件高速缓存模式对文件进行操作的平均用时会比使用无模式高速缓存模式和异步传输模式的平均用时要快很多。这里异步传输模式和无高速缓存模式的速度差不多是因为这里主要调用文件操作的函数，若能增加更多与文件操作无关的操作，异步传输模式会更有优势。

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**选作：**探究如何设计程序可以更好发挥出异步传输的性能，并尝试实现。

参考提示：设计一个函数int filter(char source，char \*sink，int f)，其中：

1、source：源文件，即从哪个文件读。

2、sink：目标文件，即写到哪个文件。

3、f：一个与文件内容无关的操作（比如空循环）。

a)仍然设计三种模式的filter函数：无缓存模式、文件高速缓存模式和异步模式。

b)给出10个大小相同内容不同的源文件，分别调用三种模式的filter函数10次，读出文件的内容，并写到10个不同的目标文件中去。

c)在调用filter函数的过程中，加入一些与文件内容无关的操作，在此情况下观察三种模式之间的用时区别（与文件无关的操作耗时不能太短，否则效果将不明显）。

应该注意的是：在调用filter函数时，要加入与文件内容无关的f操作；由于用了10个不同的源文件，对每种传输模式，可以考虑创建10个线程来并发地执行文件传输。

d)记录每次函数调用的时间，以及10次操作的总时间，分析实习结果，从中体会异步传输模式的优越性，写出心得体会。

请描述你所做的工作：

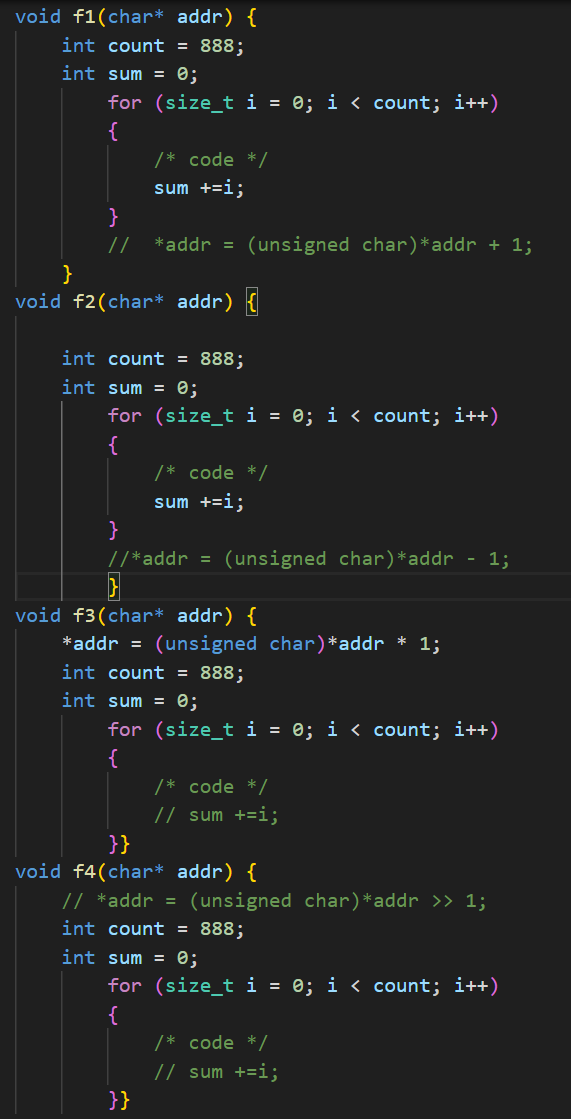


图片 2 编写nobuffer多线程处理文件函数，将每次对文件处理单独作为一个线程处理

文本

描述已自动生成

图片 3 编写sequen和overlp多线程处理文件函数，将每次对文件处理单独作为一个线程处理



图片 4 对f1234函数做修改，减少文件操作比重

文本

描述已自动生成

图片 5 对f5函数修改，减少文件操作比重

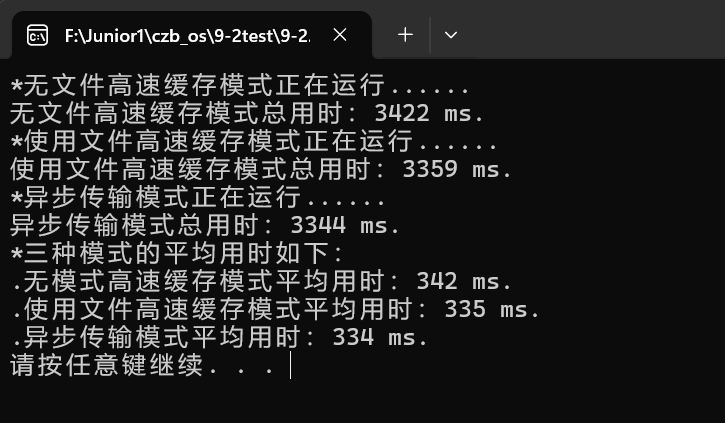
图形用户界面, 文本

描述已自动生成

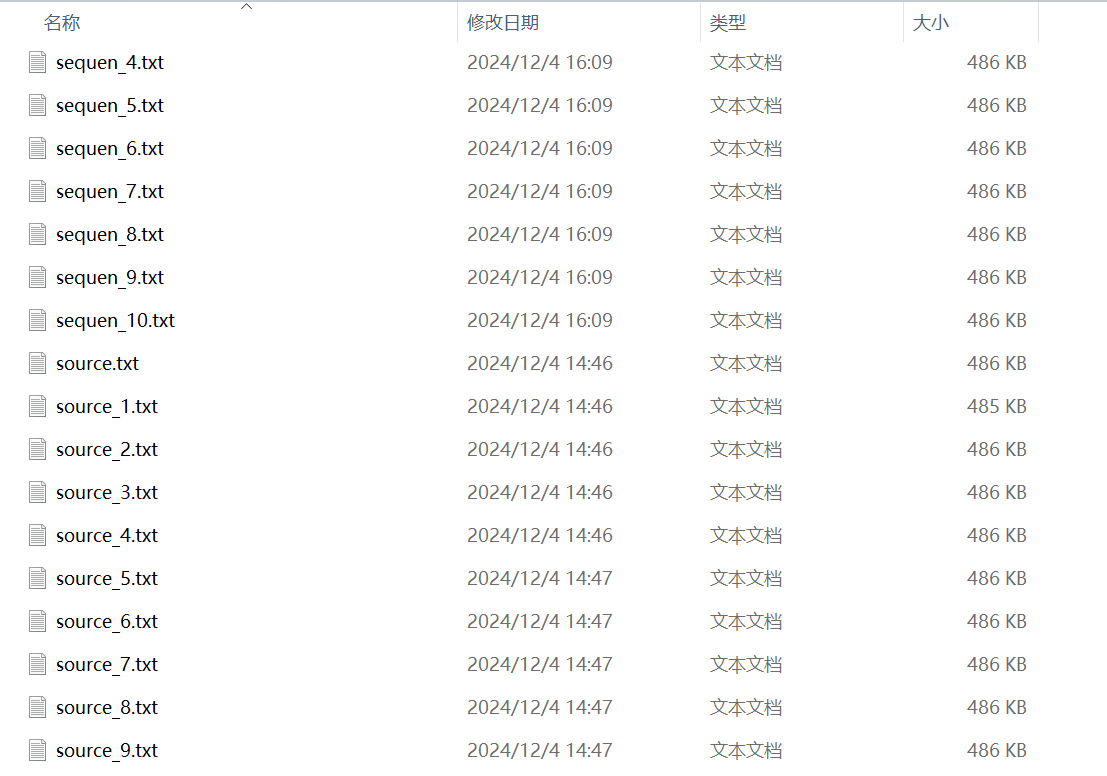
图片 6 主函数写明输入输出文件对



图片 7 主函数调用文件操作函数，并记录时间



图片 8 运行结果



图片 9 源文件一览

实验分析与心得体会

**无缓存模式**  
直接操作硬盘，文件读写速度慢。由于增加了耗时操作，这种模式的性能可能下降更多。

**顺序扫描模式**  
缓存优化后，性能有所提升。适合大文件处理，但仍然受到同步机制的限制。

**异步模式**  
异步操作减少了等待时间，与耗时操作并发运行，表现出显著优势。

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_