**C++大作业**

**基于内存缓冲区的流媒体数据缓存排序**

**姓名：张冠巍**

**学号：1130310616**

**班级：1303106**

1. **题目内容**
   1. 针对一个流媒体节目的多用户（有重复）、多线程（多区块）下载进行处理；
   2. 节目数据包有丢失、有乱序，乱序范围、丢失数量均无限定；
   3. 将收到的数据包直接写入缓冲区，缓冲区长度限定<512KB；
   4. 排好序的数据如果超过一定长度，如30KB（可设定），则输出到文件；
   5. 如果结束，将缓冲区中连续的数据写入文件；
   6. 针对多个（不确定）流媒体节目的多用户（有重复）、多线程（多区块）下载进行处理；
   7. 构造节目类，进行节目数据管理（包括缓冲区）；
   8. 构造全局的缓冲区池类：对整个系统的所有缓冲区进行管理，包括：缓冲区的申请、释放、分配、回收等（用到类模板）。
   9. 缓冲区类：实现空白区块信息（记录丢失数据包的位置）的记录。
2. **问题描述**

该问题主要有以下几个关键点：

1. **有丢失：**

数据的丢失可理解为在接收数据的过程中该数据包并未传入缓冲区，但由于事先不可能知道哪些数据会丢失，因此问题可归结于对缓冲区内已有连续数据的判定问题——判定一段数据因丢失数据而无法到达规定的输出长度，导致留在缓冲区中无法输出，同时还要避免对等待后续数据而留在缓冲区的数据进行错误的判断。

1. **有乱序：**

接收的数据段是乱序的，因此必须通过数据段在文件中的偏移量来确定各数据段的先后顺序已达到排序的目的。另外需要注意若当前接收的数据可能无法放入当前缓冲区（距当前缓冲区中的数据位置太远），以及当前接收数据需要插入到当前缓冲区中当前数据之前的情况。

1. **多用户：**

针对一个流媒体节目，多用户可以理解为在浏览器的不同标签页下播放同一个的流媒体节目，且播放进度可能是任何位置，这样就会使得接收的数据出现重复，因此多用户的问题便可视为对重复数据的处理问题。（当然即使重复处理，对程序的正确性本身并没有太大影响，但会浪费很多时间，因此主要还应综合考虑代码效率、时空复杂度等问题）

1. **多线程：**

考虑使用迅雷下载资源的情况，将流媒体数据分为不同区块同时进行操作，因此“多线程”即可考虑为“多区块”，一个缓冲区无法胜任这一工作，因此一个节目需要多个缓冲区同时进行操作从而解决多线程问题

1. **多节目：**

多个节目彼此独立，因此只要实现了对于单个节目的各种操作并封装成类，多节目问题可以看成对节目类的管理

1. **缓冲池：**

由若干缓冲区组成，将系统内缓冲区统一管理，变专用为通用。当某进程需要使用缓冲区时，提出申请，有管理程序分配给它，用完后释放缓冲区。这样可用少量缓冲区为更多进程服务。

1. **空白信息：**

由于无法预知丢失的数据，因此只有在缓冲区中的数据无法达到规定长度、确定该部分发生数据丢失后才可进行记录。

1. **方法设计**
2. **缓冲区类设计：**

**文件中的数据**

**缓冲区内连续数据**

**正在读入的数据**

**ibytes**

**m\_offset**

**tail**

**head**

**文件起始**

**StreamBuffer**

**offset**

缓冲区StreamBuffer中： 读入数据的参数：

m\_offset：缓冲区数据在文件中的偏移量 offset：读入数据在缓冲区的偏移量

head：当前连续数据头部 ibytes：读入数据的字节数

tail：当前连续数据尾部

缓冲区结构如图所示。每个缓冲区类都包含一个缓冲区及其相关操作，用于处理某片段数据的缓冲、排序工作，其主要功能如下：

* 1. **接收数据。**

缓冲区类接受数据后先判定缓冲区的剩余是否足够以及接收数据的位置，若缓冲区剩余位置不足以接收则将缓冲区中数据前移；若接收的数据在文件中的位置比缓冲区内首段连续数据的位置靠前，则将缓冲区内的数据后移。然后将数据写入缓冲区，写入后缓冲区内数据超过80%，而首段连续数据仍未达到规定的长度，则放弃该段数据，将缓冲区内其他数据前移。

* 1. **获取缓冲区中排好序的数据长度、第一个字节的偏移量和数据指针。**

对于已排好序的数据，将已排好序的数据长度tail-head、第一个字节的偏移量m\_offset + (head - m\_pData)以及数据指针head返回给外部，用于文件输出。

* 1. **删除已用过的数据。**

通过后移head指针已达到删除的目的。

* 1. **排空缓冲区**

在缓冲区任务结束后，对于缓冲区中残余的数据，全部按顺序排好并写入目标文件对应中。

* 1. **排序：**

本程序利用了优先队列实现，将各数据段的文件偏移量和数据段长度作为元素对加入优先队列进行排序，并按要求弹出当前数据中文件偏移量最小的元素对以达到排序的目的。

1. **节目类设计：**

节目类用来满足针对同一节目进行的操作，节目类中包含节目的id信息以及可能的多个缓冲区用以进行数据的接收、写入和清理。

* 1. **接收数据**

节目类接收一段数据后，现在已有的缓冲区中查询能存储这段数据，若能，则存入该缓冲区；否则再分配一个缓冲区给节目类用于存储该数据。

* 1. **写入文件**

遍历缓冲区，将满足输出条件的数据段输出后删除

* 1. **清理工作**

排空各缓冲区，并清理动态内存防止内存泄露

1. **缓冲池设计：**

缓冲池用来对系统的缓冲区进行管理。

* 1. **申请**

一次申请若干个缓冲区，准备为系统共用

* 1. **分配**

某节目需要新的缓冲区时，从备用的缓冲区中选择还未被使用的缓冲区分配给该节目。

* 1. **回收**

节目的某缓冲区被排空不再需要时，将缓冲区收回备用

* 1. **释放**

程序结束前释放所有申请的缓冲区，防止内存泄露

1. **模块设计：**
   1. **StreamBuffer缓冲区类**
      1. int ReceiveDate(unsigned int offset, unsigned int bytes, char \*pData)

功能：接受并将数据存入缓冲区

参数：

offset：数据在文件中的偏移量

bytes：存入的数据字节数

pData：指向数据的指针

返回值：存入数据的字节数

* + 1. int ContinueBytes(unsigned int &iDataOffset, char\* &pData)

功能：获取缓冲区中排好序的数据长度、第一个字节的偏移量和数据指针

参数：

iDataOffset：用于返回排好序的数据块中第一个字节的偏移量数值

pData：数据指针

返回值：缓冲区中排好序的数据长度

* + 1. int RemoveData(int iBytes);

功能：从缓冲区中删除数据

参数：

iBytes：删除的字节数

返回值：删除的字节数

* + 1. void ClearData(FILE \*fpDstFile);

功能：排空剩余数据

参数：fpDstFile指向目标文件的指针

* 1. **Program节目类**
     1. void Receive(unsigned int offset, char \*pData)

功能：接收并将数据存入缓冲区

参数：

offset：数据在文件中的偏移量

pData：指向数据的指针

* + 1. void Write(FILE \*fpDstFile)

功能：将数据写入文件

参数：

fpDstFile：要写入的文件地址

* + 1. void Clean(FILE \*fpDstFile)

功能：清理工作，将缓冲区剩余文件写入文件并删除缓冲区

参数：

fpDstFile：要写入的文件地址

* 1. **BufferPool缓冲池类**
     1. void Setup(int num)

功能：申请建立新的缓冲区

参数：

num：新建缓冲区的个数

* + 1. char\* Distribute(int id)

功能：给节目id分配一个缓冲区

参数：

id :申请缓冲区的节目id

返回值：分配缓冲区的地址

* + 1. void Recycle(int no, char\* pData)

功能：回收缓冲区

参数：

no：缓冲区的序号

pData：原来指向被回收缓冲区的指针

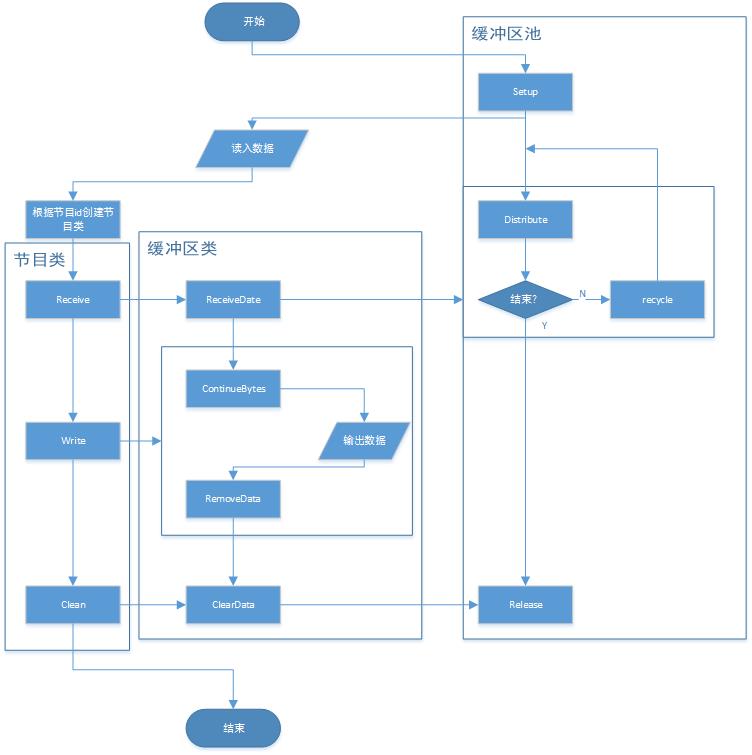
* + 1. void Release(int no)

功能：释放缓冲区

参数：

no：缓冲区的序号

1. **流程图：**



1. **源代码**

Program.h

#include <string.h>

#include "StreamBuffer.h"

struct BufferList

{

StreamBuffer \*buff;

BufferList \*next;

};

class Program

{

public:

Program(int id=0);

~Program();

void Receive(unsigned int offset, char \*pData);//节目接收数据

void Write(FILE \*fpDstFile); //将数据写入文件

void Clean(FILE \*fpDstFile);//清理工作，将缓冲区剩余文件写入文件并删除缓冲区

private:

BufferList \*l\_buffer;

int m\_id;

};

Program.cpp

#include "Program.h"

#include <string.h>

Program::Program(int id)

{

m\_id = id;

l\_buffer = new BufferList;

l\_buffer->buff = NULL;

l\_buffer->next = NULL;

}

Program::~Program()

{

delete l\_buffer;

}

/\*

功能：接收并将数据存入缓冲区

参数：

offset：数据在文件中的偏移量

pData：指向数据的指针

\*/

void Program::Receive(unsigned int offset, char \*pData)

{

if (l\_buffer->buff == NULL)//添一段，初始的情况下

{

l\_buffer->buff = new StreamBuffer;

l\_buffer->next = NULL;

l\_buffer->buff->ReceiveDate(offset, strlen(pData), pData);

}

else

{

BufferList \*temp;//用来寻找合适的缓冲区

while (temp->buff != NULL)

{

if (offset >= temp->buff->GetOffset()

&& offset+strlen(pData) < temp->buff->GetOffset()+temp->buff->GetOffset())

{//当前缓冲区能够容纳下数据

temp->buff->ReceiveDate(offset, strlen(pData), pData);

break;

}

else

{

if (temp->next != NULL)

temp = temp->next;

else//到最后一个缓冲区仍未找到合适的，则创建一个新的缓冲区

{

temp->next = new BufferList;

temp = temp->next;

temp->buff = new StreamBuffer;

temp->buff->ReceiveDate(offset, strlen(pData), pData);

temp->next = NULL;

break;

}//end for if

}//end for if

}//end for while

}//end for if

}

/\*

功能：将数据写入文件

参数：要写入的文件地址

\*/

void Program::Write(FILE \*fpDstFile)

{//这段基本照着原来的框架写……遍历输出满足条件的数据后删除

char \*pOutData;

unsigned int iOutDataOffset;

int iContinueBytes;

int iUseBytes;

BufferList \*temp = l\_buffer;

while (temp->buff != NULL)

{

iContinueBytes = temp->buff->ContinueBytes(iOutDataOffset, pOutData);

iUseBytes = iContinueBytes - 100;//假设用了一部分

if (iContinueBytes > 1024) //示例数值，可以调整

{

fseek(fpDstFile, iOutDataOffset, SEEK\_SET);

fwrite(pOutData, iUseBytes, 1, fpDstFile);

temp->buff->RemoveData(iUseBytes);

}

temp = temp->next;

}

}

/\*

功能：清理工作，将缓冲区剩余文件写入文件并删除缓冲区

参数：要写入的文件地址

\*/

void Program::Clean(FILE \*fpDstFile)

{

BufferList \*temp = l\_buffer;

while (temp->buff != NULL)

{

temp->buff->ClearData(fpDstFile);//清理缓冲区

delete temp->buff;//删除节点

temp = temp->next;

}

}

StreamBuffer.h

#if !defined SREAM\_BUFFER\_H\_HIT\_CPP

#define SREAM\_BUFFER\_H\_HIT\_CPP

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <cstring>

#include <queue>

#include <utility>

using namespace std;

class StreamBuffer

{

public:

StreamBuffer();

StreamBuffer(int iLen);

~StreamBuffer();

int ReceiveDate(unsigned int offset, unsigned int bytes, char \*pData);//接收数据

int ContinueBytes(unsigned int &iDataOffset, char\* &pData);//获取缓冲区中排好序的数据长度、第一个字节的偏移量和数据指针

int RemoveData(int iBytes);//删除 用过的数据

void ClearData(FILE \*fpDstFile);//排空剩余数据

unsigned int GetOffset();//用来给program获取缓冲区当前在文件中的偏移量

int GetBufferLen();//用来给program获取缓冲区总长度

private:

char \*m\_pData;//存数据的buffer，也可以用指针

int m\_iBufferLen;//缓冲区长度

unsigned int m\_offset;//缓冲区数据在文件中的偏移量

char \*head, \*tail; // head、tail为当前连续数据的首尾指针

priority\_queue <pair <unsigned int, unsigned int>, vector<pair <unsigned int, unsigned int> >, greater<pair <unsigned int, unsigned int> > > packInfo;

//跟大神学的……

// 使用优先队列记录数据包信息，队列中元素为pair型

// pair中两个数据分别为数据包偏移量、数据包大小

// 数据包偏移量最小的数据始终在队首(优先队列特点)

};

#endif

StreamBuffer.cpp

#include <iostream>

#include <fstream>

#include <stdio.h>

#include <stdlib.h>

#include "StreamBuffer.h"// for Code::Block

using namespace std;

//-------------------

StreamBuffer::StreamBuffer()

{

// to do： some initialization work

m\_pData = head = tail = new char[512\*1024];

m\_iBufferLen = 512\*1024;

m\_offset = 0;

}

StreamBuffer::StreamBuffer(int iLen)

{

// to do： some initialization work

m\_pData =head = tail = new char[iLen];

m\_iBufferLen = iLen;

m\_offset = 0;

}

StreamBuffer::~StreamBuffer()

{

// to do： some initialization work

if( NULL != m\_pData)

delete []m\_pData;

}

/\*

功能：接收并将数据存入缓冲区

参数：

offset：数据在文件中的偏移量

bytes：存入的数据字节数

pData：指向数据的指针

返回值：存入数据的字节数

\*/

int StreamBuffer::ReceiveDate(unsigned int offset, unsigned int bytes, char \*pData)

{

unsigned int iBytes = 0;

//若缓冲区空间不够则将数据前移，因为此时认为head之前数据已经排空，故不涉及丢包

if (offset + bytes > m\_offset + m\_iBufferLen)

{

memcpy(m\_pData, head, m\_iBufferLen - (head - m\_pData));

tail -= head - m\_pData;

m\_offset += head - m\_pData;//修正偏移量

head = m\_pData;

}

if (offset<m\_offset)//更正实验二中的错误，offset<m\_offset时把数据放在缓冲区头部

{

for (int i=m\_iBufferLen-bytes; i>=0; i--)

m\_pData[i+bytes] = m\_pData[i];

memcpy(m\_pData,pData,bytes);

head = m\_pData;

tail = m\_pData + bytes;

packInfo.push(make\_pair(offset,bytes));

}

else

{

for ( ; iBytes<bytes; iBytes++)

m\_pData[iBytes+offset-m\_offset] = pData[iBytes];//放入缓冲区

packInfo.push(make\_pair(offset, bytes)); // 将数据包信息存入优先队列

if (bytes+offset-m\_offset > m\_iBufferLen\*0.8)//超过缓存区一定部分而头部连续数据还未写入就放弃

{

//输出丢包信息，依然是head之前已经排空所以只有head和下一个head之间算是丢包

ofstream errlog("err.log",ios::app);

errlog << m\_offset+head-m\_pData << " to " << packInfo.top().first << " was lost!" << endl;

head = m\_pData - m\_offset + packInfo. top().first;//下一段连续数据的开始位置

tail = head + packInfo.top().second;//下一段连续数据的末尾

packInfo.pop();//弹出队首元素

}

}

return iBytes;// bytes the buffer saved

}

/\*

功能：获取缓冲区中排好序的数据长度、第一个字节的偏移量和数据指针

参数：

iDataOffset：用于返回排好序的数据块中第一个字节的偏移量数值

pData:数据指针

返回值：缓冲区中排好序的数据长度

\*/

int StreamBuffer::ContinueBytes(unsigned int &iDataOffset, char\* &pData)

{

int iContinueBytes = 0;

// 优先队列的队首元素记录队列中偏移量最小的数据包信息

// 当前连续部分和优先队列队首所记录的数据包连续时，更新当前连续部分，并弹出队首元素

while ((m\_offset + (tail - m\_pData)) == packInfo.top().first)

{

tail += packInfo.top().second;

packInfo.pop();

}

iDataOffset = m\_offset + (head - m\_pData);//修整偏移量

pData = head;

iContinueBytes = tail-head;//连续部分长度，即首尾指针指向的长度

return iContinueBytes;

};

/\*

功能：从缓冲区中删除数据

参数：

iBytes：删除的字节数

返回值：删除的字节数

\*/

int StreamBuffer::RemoveData(int iBytes)

{

int iBytesRemoved=0;

//从缓冲区中把数据"删除"

iBytesRemoved=iBytes;

head += iBytes;

return iBytesRemoved;

};

/\*

功能：排空剩余数据

参数：指向目标文件的指针fpDstFile

\*/

void StreamBuffer::ClearData(FILE \*fpDstFile)

{

while (!packInfo.empty())

{

//输出丢包信息

ofstream errlog("err.log",ios::app);

errlog << m\_offset+head-m\_pData << " to " << packInfo.top().first << " was lost!" << endl;

head = m\_pData - m\_offset + packInfo. top().first;//剩余连续数据的开始位置

fseek(fpDstFile,packInfo.top().first,SEEK\_SET);

fwrite(head,packInfo.top().second,1,fpDstFile);

packInfo.pop();

}

}

/\*

返回缓冲区当前在文件中的偏移量

\*/

unsigned int StreamBuffer::GetOffset()

{

return m\_offset;

}

/\*

返回缓冲区总长度

\*/

int StreamBuffer::GetBufferLen()

{

return m\_iBufferLen;

}

BufferPool.h

#include "StreamBuffer.h"

#define N 30

class BufferPool

{

public:

BufferPool(int num=10);

~BufferPool();

void Setup(int num);//建立新的缓冲区

char\* Distribute(int id);//分配缓冲区

void Recycle(int no, char\* pData);//回收缓冲区

void Release(int no);//释放缓冲区

private:

int m\_num;//缓冲区个数

char \*m\_pData[N];//备用

int m\_toID[N];//被分配到的节目id

};

BufferPool.cpp

#include "BufferPool.h"

BufferPool::BufferPool(int num)

{

m\_num = num;

for (int i=0; i<num; i++)

{

m\_pData[i] = new char[512\*1024];

m\_toID[i] = 0;

}

}

BufferPool::~BufferPool()

{

for (int i=0; i<m\_num; i++)

delete []m\_pData[i];

}

/\*

功能：申请建立新的缓冲区

参数：

num：新建缓冲区的个数

\*/

void BufferPool::Setup(int num)

{

for ( ; m\_num<num; m\_num++)

{

m\_pData[m\_num] = new char[512\*1024];

m\_toID[m\_num] = 0;

}

}

/\*

功能：给节目id分配一个缓冲区

参数：

id :申请缓冲区的节目id

返回值：分配缓冲区的地址

\*/

char\* BufferPool::Distribute(int id)

{

for (int i=0; i<m\_num; i++)

{

if (m\_toID[i] == 0)

{

m\_toID[i] = id;

return m\_pData[i];

}

}

Setup(1);

return m\_pData[m\_num];

}

/\*

功能：回收缓冲区

参数：

no：缓冲区的序号

pData：原来指向被回收缓冲区的指针

\*/

void BufferPool::Recycle(int no, char\* pData)

{

m\_toID[no] = 0;

pData = NULL;

}

/\*

功能：释放缓冲区

参数：

no：缓冲区的序号

\*/

void BufferPool::Release(int no)

{

m\_toID[no] = 0;

delete []m\_pData[no];

}