

sep 24, 19 16:48

Writer.h

Page 1/1

```

1  #ifndef _WRITER_H_
2  #define _WRITER_H_
3
4  #include <vector>
5  #include "ProtectedBitBlockQueue.h"
6  #include "OutFile.h"
7
8  class Writer {
9  private:
10     std::vector<ProtectedBitBlockQueue> vectorOfQueues;
11     OutFile* outFile;
12     unsigned currentQueue;
13     int amountOfQueues;
14 public:
15     explicit Writer(OutFile* outFile, int maxElements, int amountOfQueues);
16
17     void write();
18
19     ProtectedBitBlockQueue* getQueueFor(int thread);
20
21     void nextQueue();
22
23     ~Writer();
24 };
25
26 #endif

```

sep 24, 19 16:48

Writer.cpp

Page 1/1

```

1  #include "Writer.h"
2  #define FIRST_QUEUE 0
3
4  Writer::Writer(OutFile *outFile, int maxElements, int amountOfQueues) {
5      for (int i = 0; i < amountOfQueues; i++){
6          vectorOfQueues.push_back(ProtectedBitBlockQueue(maxElements));
7      }
8      this → outFile = outFile;
9      this → amountOfQueues = amountOfQueues;
10     currentQueue = FIRST_QUEUE;
11 }
12
13 void Writer::write() {
14     while (!vectorOfQueues.at(currentQueue).donePopping()) {
15         BitBlock bitBlock = std::move(vectorOfQueues.at(currentQueue).pop());
16         bitBlock.writeTo(outFile);
17         nextQueue();
18     }
19 }
20
21 ProtectedBitBlockQueue* Writer::getQueueFor(int thread) {
22     return &vectorOfQueues.at(thread);
23 }
24
25 void Writer::nextQueue() {
26     unsigned newQueue = currentQueue + 1;
27     if (newQueue < vectorOfQueues.size()) {
28         currentQueue = newQueue;
29     } else {
30         currentQueue = FIRST_QUEUE;
31     }
32 }
33
34 Writer::~Writer() {
35 }

```

sep 24, 19 16:48

Thread.h

Page 1/1

```

1  #ifndef _THREAD_H_
2  #define _THREAD_H_
3
4  #include <thread>
5
6  class Thread {
7  private:
8      std::thread thread;
9
10 public:
11     Thread();
12
13     Thread(const Thread&) = delete;
14
15     Thread& operator=(const Thread&) = delete;
16
17     Thread(Thread^ other);
18
19     Thread& operator=(Thread ^other);
20
21     virtual void run() = 0;
22
23     void start();
24
25     void join();
26
27     virtual ~Thread();
28 };
29
30
31 #endif

```

sep 24, 19 16:48

Thread.cpp

Page 1/1

```

1  #include "Thread.h"
2
3  Thread::Thread() {
4  }
5
6  void Thread::start() {
7      thread = std::thread(&Thread::run, this);
8  }
9
10 void Thread::join() {
11     thread.join();
12 }
13
14 Thread::Thread(Thread ^other) {
15     this → thread = std::move(other.thread);
16 }
17
18 Thread& Thread::operator=(Thread ^other) {
19     this → thread = std::move(other.thread);
20     return *this;
21 }
22
23 Thread::~Thread() {
24 }

```

sep 24, 19 16:48

ProtectedInFile.h

Page 1/1

```

1  #ifndef _IN_FILE_H_
2  #define _IN_FILE_H_
3  #include <istream>
4  #include "Block.h"
5  #include <mutex>
6  #include <fstream>
7  class ProtectedInFile {
8  private:
9      std::istream* file;
10     std::ifstream fd;
11     std::mutex mutex;
12     unsigned lastRead;
13     bool wasRead;
14     int size;
15
16     int readNumberTo(Block* block);
17
18     int isEOF();
19
20 public:
21     explicit ProtectedInFile(const char* filename);
22
23     int readNumbsToStartingAt(int amountOfNumb, Block* block, int position);
24
25     ~ProtectedInFile();
26 };
27
28 #endif

```

sep 24, 19 16:48

ProtectedInFile.cpp

Page 1/2

```

1  #include "ProtectedInFile.h"
2  #include <iostream>
3  #include <cstdlib>
4  #include <cstring>
5  #include <netinet/in.h>
6  #include "Lock.h"
7  #define END_OF_FILE -1
8  #define OK 0
9  #define NUM_SIZE 4
10 #define CIN '-'
11
12 ProtectedInFile::ProtectedInFile(const char* filename) {
13     if (*filename == CIN) {
14         file = &std::cin;
15     } else {
16         fd = std::ifstream(filename, std::ios::in|std::ios::binary);
17         file = &fd;
18     }
19     lastRead = 0;
20     file → seekg(0, file → end);
21     size = file → tellg();
22     file → seekg(0, file → beg);
23 }
24
25 int ProtectedInFile::readNumbsToStartingAt(int amountOfNumb, Block *block,
26     int position) {
27     mutex.lock();
28     file → seekg(position, file → beg);
29     int fileState = OK;
30     if (file → tellg() ≥ size) {
31         fileState = END_OF_FILE;
32     }
33     wasRead = false;
34     while (block → hasSpace() ^ fileState == OK) {
35         fileState = readNumberTo(block);
36     }
37     while (block → hasSpace() ^ wasRead) {
38         block → addNumber(lastRead);
39     }
40     mutex.unlock();
41     return fileState;
42 }
43
44 int ProtectedInFile::readNumberTo(Block *block) {
45     int fileState = isEOF();
46     if (fileState == OK) {
47         char* num = new char [NUM_SIZE];
48         file → read(num, NUM_SIZE);
49         uint32_t number;
50         memcpy(&number, num, sizeof(char) * NUM_SIZE);
51         delete[] num;
52         number = ntohl(number);
53         fileState = isEOF();
54         if (fileState == OK) {
55             lastRead = number;
56             block → addNumber(lastRead);
57             wasRead = true;
58         }
59     }
60     return fileState;
61 }
62
63 int ProtectedInFile::isEOF() {
64     int returnValue = OK;
65     if (file → eof()) {
66         file → clear();

```

sep 24, 19 16:48

ProtectedInFile.cpp

Page 2/2

```

67     returnVal = END_OF_FILE;
68 }
69 return returnVal;
70 }
71
72 ProtectedInFile::~ProtectedInFile(){
73     if (fd.is_open()) {
74         fd.close();
75     }
76 }

```

sep 24, 19 16:48

ProtectedBitBlockQueue.h

Page 1/1

```

1  #ifndef _PROTECTED_BLOCK_QUEUE_H_
2  #define _PROTECTED_BLOCK_QUEUE_H_
3
4  #include <queue>
5  #include <mutex>
6  #include <condition_variable>
7  #include "BitBlock.h"
8  #include "OutFile.h"
9
10 class ProtectedBitBlockQueue {
11     private:
12         std::queue<BitBlock> queue;
13         std::mutex m;
14         std::condition_variable popCondition;
15         std::condition_variable pushCondition;
16         unsigned maxElements;
17         bool donePushing;
18         bool popAvailable;
19
20     public:
21         explicit ProtectedBitBlockQueue(int maxAmountOfElements);
22
23         ProtectedBitBlockQueue(const ProtectedBitBlockQueue& other);
24
25         void push(BitBlock bitBlock);
26
27         BitBlock pop();
28
29         bool donePopping();
30
31         void done(bool processState);
32
33         ~ProtectedBitBlockQueue();
34 };
35
36 #endif
37

```

sep 24, 19 16:48

ProtectedBitBlockQueue.cpp

Page 1/1

```

1  #include "ProtectedBitBlockQueue.h"
2  #include "Lock.h"
3
4  ProtectedBitBlockQueue::ProtectedBitBlockQueue(int maxAmountOfElements) {
5      maxElements = maxAmountOfElements;
6      donePushing = false;
7      popAvailable = false;
8  }
9
10 ProtectedBitBlockQueue::ProtectedBitBlockQueue
11     (const ProtectedBitBlockQueue &other):queue(other.queue) {
12     this → maxElements = other.maxElements;
13     this → donePushing = other.donePushing;
14     this → popAvailable = other.popAvailable;
15 }
16
17 void ProtectedBitBlockQueue::push(BitBlock bitBlock) {
18     std::unique_lock<std::mutex> lock(m);
19     while (queue.size() ≥ maxElements) {
20         pushCondition.wait(lock);
21     }
22     queue.push(std::move(bitBlock));
23     done(false);
24 }
25
26 BitBlock ProtectedBitBlockQueue::pop() {
27     std::unique_lock<std::mutex> lock(m);
28     BitBlock bitBlock(0, 0, 0);
29     while (¬popAvailable ∧ ¬donePushing) {
30         popCondition.wait(lock);
31     }
32     if (queue.size() > 0) {
33         bitBlock = std::move(queue.front());
34         queue.pop();
35         popAvailable = false;
36     }
37     pushCondition.notify_all();
38     return std::move(bitBlock);
39 }
40
41 bool ProtectedBitBlockQueue::donePopping() {
42     bool answer = false;
43     if (donePushing ∧ queue.size() == 0) {
44         answer = true;
45     }
46     return answer;
47 }
48
49 void ProtectedBitBlockQueue::done(bool processState) {
50     popAvailable = true;
51     donePushing = processState;
52     popCondition.notify_all();
53 }
54
55 ProtectedBitBlockQueue::~ProtectedBitBlockQueue() {
56 }

```

sep 24, 19 16:48

OutFile.h

Page 1/1

```

1  #ifndef _OUT_FILE_H_
2  #define _OUT_FILE_H_
3
4  #include <ostream>
5  #include <iostream>
6  #include <fstream>
7  class OutFile {
8      private:
9          std::ostream* outFile;
10         std::ofstream fd;
11
12     public:
13         explicit OutFile(const char* filename);
14
15         void write(char* buf, int bytesToWrite);
16
17         ~OutFile();
18     };
19
20
21 #endif

```

sep 24, 19 16:48

OutFile.cpp

Page 1/1

```

1  #include "OutFile.h"
2  #include <fstream>
3  #define COUT '-'
4
5  OutFile::OutFile(const char* filename) {
6      if (*filename == COUT) {
7          outFile = &std::cout;
8      } else {
9          fd = std::ofstream(filename, std::ios::binary);
10         outFile = &fd;
11     }
12 }
13
14 void OutFile::write(char *buf, int bytesToWrite) {
15     outFile->write(buf, bytesToWrite);
16 }
17
18 OutFile::~OutFile() {
19     if (fd.is_open()) {
20         fd.close();
21     }
22 }

```

sep 24, 19 16:48

main.cpp

Page 1/1

```

1  #include "ProtectedInFile.h"
2  #include "FileCompressor.h"
3  #include "OutFile.h"
4  #include "Writer.h"
5  #include <vector>
6  int main(int argc, char const *argv[]) {
7      ProtectedInFile file(argv[4]);
8      OutFile outFile(argv[5]);
9      int numbsPerBlock = atoi(argv[1]);
10     int amountOfThreads = atoi(argv[2]);
11     int elementsPerQueue = atoi(argv[3]);
12     Writer writer(&outFile, elementsPerQueue, amountOfThreads);
13     std::vector<FileCompressor*> threads;
14     for (int i = 0; i < amountOfThreads; i++) {
15         threads.push_back(new FileCompressor(&file, writer.getQueueFor(i),
16                                             numbsPerBlock, amountOfThreads, i));
17         threads[i] -> start();
18     }
19     writer.write();
20     for (int i = 0; i < (*argv[2] - '0'); i++) {
21         threads[i] -> join();
22         delete threads[i];
23     }
24     return 0;
25 }

```

sep 24, 19 16:48

Lock.h

Page 1/1

```
1  #ifndef TP2_LOCK_H
2  #define TP2_LOCK_H
3
4  #include <mutex>
5
6  class Lock {
7  private:
8      std::mutex& mutex;
9  public:
10     explicit Lock(std::mutex& aMutex);
11
12     ~Lock();
13 };
14 #endif
```

sep 24, 19 16:48

Lock.cpp

Page 1/1

```
1  #include "Lock.h"
2
3  Lock::Lock(std::mutex &aMutex) :mutex(aMutex) {
4      mutex.lock();
5  }
6
7  Lock::~~Lock() {
8      mutex.unlock();
9  }
```

sep 24, 19 16:48

FileCompressor.h

Page 1/1

```

1  #ifndef _FILE_COMPRESSOR_H_
2  #define _FILE_COMPRESSOR_H_
3  #include "ProtectedInFile.h"
4  #include "ProtectedBitBlockQueue.h"
5  #include "Block.h"
6  #include "Thread.h"
7  class FileCompressor : public Thread {
8  private:
9      ProtectedInFile* inFile;
10     ProtectedBitBlockQueue* queue;
11     Block block;
12     int numbsPerBlock;
13     int numbOfThreads;
14     int myNumb;
15
16 public:
17     explicit FileCompressor(ProtectedInFile* in, ProtectedBitBlockQueue* queue,
18         int numbsPerBlock, int numbOfThreads, int myNumb);
19
20     void run();
21
22     ~FileCompressor();
23 };
24
25 #endif

```

sep 24, 19 16:48

FileCompressor.cpp

Page 1/1

```

1  #include "FileCompressor.h"
2  #define END_OF_FILE -1
3  #define OK 0
4  #define NUMB_SIZE 4
5
6  FileCompressor::FileCompressor(ProtectedInFile* in, ProtectedBitBlockQueue*
7      queue, int numbsPerBlock, int numbOfThreads, int myNumb)
8      :block(numbsPerBlock) {
9      inFile = in;
10     this → queue = queue;
11     this → numbsPerBlock = numbsPerBlock;
12     this → numbOfThreads = numbOfThreads;
13     this → myNumb = myNumb;
14 }
15
16 void FileCompressor::run() {
17     int fileState = OK;
18     int pos = (numbsPerBlock * NUMB_SIZE * myNumb);
19     while (fileState == OK) {
20         fileState = inFile → readNumbsToStartingAt(numbsPerBlock, &block, pos);
21         pos += (numbsPerBlock * NUMB_SIZE * numbOfThreads);
22         if (¬(block.hasSpace())) {
23             block.compressTo(queue);
24             block.reset();
25         }
26     }
27     queue → done(true);
28 }
29
30 FileCompressor::~FileCompressor() {
31 }

```


sep 24, 19 16:48

Block.h

Page 1/1

```

1  #ifndef _BLOCK_H_
2  #define _BLOCK_H_
3  #include <vector>
4  #include "BitBlock.h"
5  #include "ProtectedBitBlockQueue.h"
6  #include "OutFile.h"
7
8  class Block {
9  private:
10     unsigned minNumb;
11     unsigned maxNumb;
12     std::vector<unsigned>::iterator iterator;
13     std::vector<unsigned> numbs;
14     BitBlock* bits;
15
16     void updateMax(unsigned numberAdded);
17     void updateMin(unsigned numberAdded);
18
19
20
21 public:
22     explicit Block(int amountOfNumbs);
23
24     void addNumber(unsigned numbToAdd);
25
26     void reset();
27
28     bool hasSpace();
29
30     void compressTo(ProtectedBitBlockQueue* queue);
31
32     ~Block();
33 };
34 #endif

```

sep 24, 19 16:48

Block.cpp

Page 1/1

```

1  #include "Block.h"
2  #include "BitBlock.h"
3  #include "ProtectedBitBlockQueue.h"
4  #define MIN_VALUE 0
5  #define MAX_VALUE 0xffffffff
6
7  Block::Block(int amountOfNumbs) {
8      numbs.resize(amountOfNumbs);
9      reset();
10 }
11
12 void Block::addNumber(unsigned numbToAdd) {
13     if (iterator < numbs.end()) {
14         updateMin(numbToAdd);
15         updateMax(numbToAdd);
16         *iterator = numbToAdd;
17         iterator ++;
18     }
19 }
20
21 void Block::updateMax(unsigned numberAdded) {
22     if (maxNumb < numberAdded) {
23         maxNumb = numberAdded;
24     }
25 }
26
27 void Block::updateMin(unsigned numberAdded) {
28     if (minNumb > numberAdded) {
29         minNumb = numberAdded;
30     }
31 }
32 void Block::reset() {
33     iterator = numbs.begin();
34     minNumb = MAX_VALUE;
35     maxNumb = MIN_VALUE;
36 }
37
38 bool Block::hasSpace() {
39     bool answer = false;
40     if (iterator < numbs.end()) {
41         answer = true;
42     }
43     return answer;
44 }
45
46 void Block::compressTo(ProtectedBitBlockQueue *queue) {
47     BitBlock bitBlock(minNumb, (maxNumb - minNumb), numbs.size());
48     for (iterator = numbs.begin(); iterator < numbs.end(); iterator++) {
49         bitBlock.addNumb(*iterator - minNumb);
50     }
51     bitBlock.addPadding();
52     queue → push(std::move(bitBlock));
53 }
54
55 Block::~Block() {
56 }

```

sep 24, 19 16:48

BitBlock.h

Page 1/1

```

1  #ifndef _BIT_BLOCK_H_
2  #define _BIT_BLOCK_H_
3  #include "OutFile.h"
4  #include <vector>
5  class BitBlock {
6  private:
7      unsigned reference;
8      std::vector<char>::iterator iterator;
9      std::vector<char> bytes;
10     char aux;
11     int inBit;
12     bool validBlock;
13     unsigned bitsPerNumb;
14
15     void nextBit();
16     unsigned calculateBitsPerNumb(unsigned maxNumb);
17     int calculateBytesNeeded(int amountOfNumbs);
18
19 public:
20     explicit BitBlock(unsigned aReference, unsigned maxNumb, int amountOfNumbs);
21
22     BitBlock(const BitBlock& other) = default;
23
24     BitBlock(BitBlock^ other);
25
26     BitBlock&operator=(BitBlock^ other);
27
28     void addNumb(unsigned numbToAdd);
29
30     void writeTo(OutFile *outFile);
31
32     void addPadding();
33
34     ~BitBlock();
35 };
36 #endif

```

sep 24, 19 16:48

BitBlock.cpp

Page 1/2

```

1  #include "BitBlock.h"
2  #include <iostream>
3  #include <bitset>
4  #include "OutFile.h"
5  #include <netinet/in.h>
6  #include <bitset>
7  #define MAX_BIT_QUANTITY 32
8  #define FIRST_BIT 7
9  #define LAST_BIT 0
10 #define REFERENCE_SIZE 4
11
12 BitBlock::BitBlock(unsigned aReference, unsigned maxNumb, int amountOfNumbs) {
13     bitsPerNumb = calculateBitsPerNumb(maxNumb);
14     bytes.resize(calculateBytesNeeded(amountOfNumbs));
15     iterator = bytes.begin();
16     inBit = FIRST_BIT;
17     reference = aReference;
18     validBlock = true;
19     if (amountOfNumbs == 0) {
20         validBlock = false;
21     }
22 }
23
24 BitBlock::BitBlock(BitBlock ^other): iterator(std::move(other.iterator)),
25     bytes(std::move(other.bytes)) {
26     this → reference = other.reference;
27     this → aux = other.aux;
28     this → bitsPerNumb = other.bitsPerNumb;
29     this → inBit = other.inBit;
30     this → validBlock = other.validBlock;
31 }
32
33 BitBlock& BitBlock::operator=(BitBlock ^other) {
34     if (this == &other) {
35         return *this;
36     }
37     this → iterator = std::move(other.iterator);
38     this → bytes = std::move(other.bytes);
39     this → reference = other.reference;
40     this → aux = other.aux;
41     this → bitsPerNumb = other.bitsPerNumb;
42     this → inBit = other.inBit;
43     this → validBlock = other.validBlock;
44     return *this;
45 }
46
47 int BitBlock::calculateBytesNeeded(int amountOfNumbs) {
48     int amountOfBits = amountOfNumbs * bitsPerNumb;
49     int bytesNeeded = amountOfBits / 8;
50     if ((amountOfBits % 8) != 0) {
51         bytesNeeded ++;
52     }
53     return bytesNeeded;
54 }
55
56 unsigned int BitBlock::calculateBitsPerNumb(unsigned maxNumb) {
57     std::bitset<MAX_BIT_QUANTITY> bits(maxNumb);
58     int inBit = 0;
59     unsigned index = MAX_BIT_QUANTITY;
60     if (maxNumb != 0) {
61         while (inBit == 0 ^ index > 0) {
62             index --;
63             inBit = bits[index];
64         }
65         index ++;
66     } else {

```

sep 24, 19 16:48

BitBlock.cpp

Page 2/2

```

67     index = 0;
68 }
69 return index;
70 }
71
72 void BitBlock::addNumb(unsigned numbToAdd) {
73     std::bitset<MAX_BIT_QUANTITY> source(numbToAdd);
74     int amountOfBits = bitsPerNumb;
75     while (0 < amountOfBits) {
76         aux = (aux &~ (1UL << inBit)) | (source[amountOfBits - 1] << inBit);
77         nextBit();
78         amountOfBits --;
79     }
80 }
81
82 void BitBlock::writeTo(OutFile* outFile) {
83     if (validBlock) {
84         reference = ntohl(reference);
85         outFile → write((char*) &reference, REFERENCE_SIZE);
86         outFile → write((char*) &bitsPerNumb, 1);
87         for (iterator = bytes.begin(); iterator < bytes.end(); iterator++) {
88             outFile → write(&(*iterator), 1);
89         }
90     }
91 }
92
93 void BitBlock::addPadding() {
94     if (iterator ≠ bytes.end() ^ inBit ≠ FIRST_BIT) {
95         while (inBit ≥ 0) {
96             aux = (aux &~ (1UL << inBit)) | (0 << inBit);
97             inBit --;
98         }
99         *iterator = aux;
100     }
101 }
102
103 void BitBlock::nextBit() {
104     if (inBit == LAST_BIT) {
105         inBit = FIRST_BIT;
106         if (iterator ≠ bytes.end()){
107             *iterator = aux;
108             iterator ++;
109         }
110     } else {
111         inBit --;
112     }
113 }
114
115 BitBlock::~BitBlock() {
116 }

```

sep 24, 19 16:48

Table of Content

Page 1/1

1	Table of Contents					
2	1	Writer.h.....	sheets	1 to	1 (1) pages	1- 1 28 lines
3	2	Writer.cpp.....	sheets	1 to	1 (1) pages	2- 2 36 lines
4	3	Thread.h.....	sheets	2 to	2 (1) pages	3- 3 32 lines
5	4	Thread.cpp.....	sheets	2 to	2 (1) pages	4- 4 25 lines
6	5	ProtectedInFile.h...	sheets	3 to	3 (1) pages	5- 5 29 lines
7	6	ProtectedInFile.cpp.	sheets	3 to	4 (2) pages	6- 7 77 lines
8	7	ProtectedBitBlockQueue.h	sheets	4 to	4 (1) pages	8- 8 38 lines
9	8	ProtectedBitBlockQueue.cpp	sheets	5 to	5 (1) pages	9- 9 57 lines
10	9	OutFile.h.....	sheets	5 to	5 (1) pages	10- 10 22 lines
11	10	OutFile.cpp.....	sheets	6 to	6 (1) pages	11- 11 23 lines
12	11	main.cpp.....	sheets	6 to	6 (1) pages	12- 12 26 lines
13	12	Lock.h.....	sheets	7 to	7 (1) pages	13- 13 15 lines
14	13	Lock.cpp.....	sheets	7 to	7 (1) pages	14- 14 10 lines
15	14	FileCompressor.h....	sheets	8 to	8 (1) pages	15- 15 26 lines
16	15	FileCompressor.cpp..	sheets	8 to	8 (1) pages	16- 16 32 lines
17	16	Block.h.....	sheets	9 to	9 (1) pages	17- 17 35 lines
18	17	Block.cpp.....	sheets	9 to	9 (1) pages	18- 18 57 lines
19	18	BitBlock.h.....	sheets	10 to	10 (1) pages	19- 19 38 lines
20	19	BitBlock.cpp.....	sheets	10 to	11 (2) pages	20- 21 117 lines