

oct 01, 19 23:38

WriterThread.h

Page 1/1

```

1  #ifndef _WRITER_THREAD_H_
2  #define _WRITER_THREAD_H_
3
4  #include "ProtectedQueue.h"
5  #include "Writer.h"
6  #include <thread>
7  #include <vector>
8  #include <cstdbool>
9
10 class WriterThread {
11     Writer& writer;
12     std::thread thread;
13     std::vector<ProtectedQueue>& queues;
14
15     private:
16         void write_file();
17
18         bool queues_are_open();
19
20         bool queues_are_empty();
21
22     public:
23         explicit WriterThread(std::vector<ProtectedQueue>& queues,
24                               Writer& writer);
25
26         void run();
27
28         void join();
29
30         ~WriterThread();
31 };
32
33 #endif

```

oct 01, 19 23:38

WriterThread.cpp

Page 1/1

```

1  #include "WriterThread.h"
2  #include "ProtectedQueue.h"
3  #include "Block.h"
4  #include "Writer.h"
5  #include <vector>
6  #include <fstream>
7  #include <cstdbool>
8  #include <iostream>
9
10 /*-----Public-----*/
11 WriterThread::WriterThread(std::vector<ProtectedQueue>& queues,
12                             Writer& writer):
13     writer(writer),
14     queues(queues) {}
15
16 void WriterThread::run() {
17     this->thread = std::thread(&WriterThread::write_file, this);
18 }
19
20 void WriterThread::join() {
21     this->thread.join();
22 }
23
24 /*-----Private-----*/
25 void WriterThread::write_file() {
26     while (this->queues_are_open() & !this->queues_are_empty()) {
27         for (size_t i = 0; i < queues.size(); i++) {
28             Block* block = this->queues[i].pop();
29             if (block) {
30                 block->print_in_file(this->writer);
31                 delete block;
32             }
33         }
34     }
35 }
36
37 bool WriterThread::queues_are_open() {
38     bool open = false;
39     for (size_t i = 0; i < this->queues.size(); i++) {
40         if (!this->queues[i].closed()) {
41             open = true;
42         }
43     }
44     return open;
45 }
46
47 bool WriterThread::queues_are_empty() {
48     bool empty = true;
49     for (size_t i = 0; i < this->queues.size(); i++) {
50         if (!this->queues[i].empty()) {
51             empty = false;
52         }
53     }
54     return empty;
55 }
56
57 WriterThread::~WriterThread() {}

```

oct 01, 19 23:38

Writer.h

Page 1/1

```

1  #ifndef _WRITER_H_
2  #define _WRITER_H_
3
4  #include <fstream>
5  #include <stdint>
6
7  class Writer {
8      std::ostream* output; //Current output
9      std::ofstream file;
10
11     private:
12         uint8_t get_byte_to_print(const char* number_by_bit);
13
14     public:
15         Writer();
16
17         int set_file(const char* filename);
18
19         void write_bits(uint8_t bits);
20
21         void write_reference(uint32_t reference);
22
23         void write_number(const char* number_by_bit);
24
25         ~Writer();
26 };
27
28 #endif

```

oct 01, 19 23:38

Writer.cpp

Page 1/1

```

1  #include "Writer.h"
2  #include <fstream>
3  #include <iostream>
4  #include <stdint>
5  #include <arpa/inet.h>
6
7  #define ERROR 1
8  #define SUCCESS 0
9  #define BINARY +2
10
11 /*-----Public-----*/
12 Writer::Writer() {
13     this->output = &std::cout; //Default
14 }
15
16 int Writer::set_file(const char* filename) {
17     this->file.open(filename, std::ios::binary);
18     if (this->file.is_open()) {
19         this->output = &this->file;
20         return SUCCESS;
21     }
22     return ERROR;
23 }
24
25 void Writer::write_reference(uint32_t reference) {
26     uint32_t ref_be = htonl(reference);
27     output->write((char*)&ref_be, sizeof(uint32_t));
28 }
29
30 void Writer::write_bits(uint8_t bits) {
31     output->write((char*)&bits, sizeof(uint8_t));
32 }
33
34 void Writer::write_number(const char* number_by_bit) {
35     uint8_t number = this->get_byte_to_print(number_by_bit);
36     output->write((char*)&number, sizeof(uint8_t));
37 }
38
39 /*-----Private-----*/
40 uint8_t Writer::get_byte_to_print(const char* number_by_bit) {
41     return (uint8_t)strtoul(number_by_bit, nullptr, BINARY);
42 }
43
44 Writer::~Writer() {
45     if (this->file.is_open()) {
46         this->file.close();
47     }
48 }

```

oct 01, 19 23:38

Reader.h

Page 1/1

```

1  #ifndef _READER_H_
2  #define _READER_H_
3
4  #include "BlockBuffer.h"
5  #include <mutex>
6  #include <fstream>
7
8  class Reader {
9      std::istream* input; //Current input
10     std::ifstream file;
11     std::mutex f_mtx;
12     size_t curr_pos;
13     int block_len;
14
15     private:
16         int set_block(int block_pos);
17
18         int read_block(BlockBuffer& buffer);
19
20     public:
21         explicit Reader(int block_len);
22
23         int set_file(const char* filename);
24
25         int set_and_read_block(int block_pos, BlockBuffer& buffer);
26
27         ~Reader();
28 };
29
30 #endif

```

oct 01, 19 23:38

Reader.cpp

Page 1/1

```

1  #include "Reader.h"
2  #include "BlockBuffer.h"
3  #include <mutex>
4  #include <fstream>
5  #include <cstdlib>
6  #include <cstring>
7
8  #define ERROR 1
9  #define SUCCESS 0
10
11  /*-----Public-----*/
12  Reader::Reader(int block_len) {
13      this->input = &std::cin; //Default
14      this->block_len = block_len;
15  }
16
17  int Reader::set_file(const char* filename) {
18      this->file.open(filename, std::ios::binary);
19      if (this->file.is_open()) {
20          this->input = &this->file;
21          return SUCCESS;
22      }
23      return ERROR;
24  }
25
26  int Reader::set_and_read_block(int block_pos, BlockBuffer& buffer) {
27      std::unique_lock<std::mutex> lock(this->f_mtx);
28      if (this->set_block(block_pos) == ERROR) {
29          return 0;
30      }
31      return this->read_block(buffer);
32  }
33
34  /*-----Private-----*/
35  int Reader::set_block(int block_pos) {
36      int pos = block_pos * this->block_len * sizeof(uint32_t);
37      if (this->input->seekg(pos, std::ios_base::beg)) {
38          return SUCCESS;
39      }
40      this->input->clear();
41      return ERROR;
42  }
43
44  int Reader::read_block(BlockBuffer& buffer) {
45      char number[DW_BYTES];
46      memset(number, 0, DW_BYTES * sizeof(char));
47
48      while (!buffer.is_full() ^ this->input->read(number, DW_BYTES)) {
49          buffer.add_number(number);
50      }
51      if (this->input->eof() ^ this->input->fail()) {
52          this->input->clear();
53      }
54
55      return buffer.numbers_stored();
56  }
57
58  Reader::~Reader() {
59      if (this->file.is_open()) {
60          this->file.close();
61      }
62  }

```

oct 01, 19 23:38

ProtectedQueue.h

Page 1/1

```

1  #ifndef _QUEUE_H_
2  #define _QUEUE_H_
3
4  #include "Block.h"
5  #include <queue>
6  #include <mutex>
7  #include <cstdbool>
8  #include <condition_variable>
9
10 class ProtectedQueue {
11     size_t max_q_len;
12     std::queue<Block*> queue;
13     std::condition_variable cv;
14     std::mutex q_mtx;
15     bool q_closed;
16
17     public:
18         explicit ProtectedQueue(size_t max_q_len);
19
20         ProtectedQueue(ProtectedQueue^ p_queue);
21
22         void push(Block* block);
23
24         bool empty();
25
26         void close();
27
28         bool closed();
29
30         Block* pop();
31
32         ~ProtectedQueue();
33 };
34
35 #endif

```

oct 01, 19 23:38

ProtectedQueue.cpp

Page 1/2

```

1  #include "ProtectedQueue.h"
2  #include "Block.h"
3  #include <cstdbool>
4  #include <iostream>
5  #include <mutex>
6
7  /*-----Public-----*/
8  ProtectedQueue::ProtectedQueue(size_t max_q_len) {
9      this->max_q_len = max_q_len;
10     this->q_closed = false;
11 }
12
13 ProtectedQueue::ProtectedQueue(ProtectedQueue^ p_queue):
14     queue(p_queue->queue)
15 {
16     this->max_q_len = p_queue->max_q_len;
17     this->q_closed = p_queue->q_closed;
18
19     p_queue->max_q_len = 0;
20     p_queue->q_closed = true;
21 }
22
23 void ProtectedQueue::push(Block* block) {
24     std::unique_lock<std::mutex> lock(this->q_mtx);
25     while (this->queue.size() ≥ this->max_q_len) {
26         this->cv.wait(lock);
27     }
28
29     this->queue.push(block);
30     this->cv.notify_all();
31 }
32
33 Block* ProtectedQueue::pop() {
34     std::unique_lock<std::mutex> lock(this->q_mtx);
35     while (this->queue.empty() ^ !this->q_closed) {
36         this->cv.wait(lock);
37     }
38
39     if (this->queue.empty()) {
40         //The queue is not receiving more elements
41         return nullptr;
42     } else {
43         Block* block = this->queue.front();
44         this->queue.pop();
45         this->cv.notify_all();
46         return block;
47     }
48 }
49
50
51 bool ProtectedQueue::empty() {
52     std::unique_lock<std::mutex> lock(this->q_mtx);
53     return this->queue.empty();
54 }
55
56 void ProtectedQueue::close() {
57     std::unique_lock<std::mutex> lock(this->q_mtx);
58     this->q_closed = true;
59     //Notify free pass to take!
60     this->cv.notify_all();
61 }
62
63 bool ProtectedQueue::closed() {
64     std::unique_lock<std::mutex> lock(this->q_mtx);
65     return this->q_closed;
66 }

```

oct 01, 19 23:38

ProtectedQueue.cpp

Page 2/2

```

67
68 /*-----Private-----*/
69 ProtectedQueue::~ProtectedQueue() {}

```

oct 01, 19 23:38

main.cpp

Page 1/1

```

1  #include "Compressor.h"
2  #include <iostream>
3  #include <cstring>
4  #include <cstdlib>
5  #include <sstream>
6
7  #define ERROR 1
8  #define SUCCESS 0
9  #define NO_FILE "-"
10
11 int main(int argc, char *argv[]) {
12     if (argc != 6) {
13         std::cerr << "Parametros invalidos" << '\n';
14         return ERROR;
15     }
16
17     char* end_1; char* end_2; char* end_3;
18     int block_len = strtol(argv[1], &end_1, 10);
19     int num_thrds = strtol(argv[2], &end_2, 10);
20     int max_q_len = strtol(argv[3], &end_3, 10);
21
22     if (*end_1 || *end_2 || *end_3) {
23         std::cerr << "Los parametros numericos de entrada son invalidos" << '\n';
24     }
25
26     Compressor compressor(num_thrds, max_q_len, block_len);
27
28     if (strcmp(NO_FILE, argv[4]) != 0) {
29         if (compressor.set_input_file(argv[4]) == ERROR) {
30             std::cerr << "El archivo no se pudo abrir o no existe" << '\n';
31             return ERROR;
32         }
33     }
34     if (strcmp(NO_FILE, argv[5]) != 0) {
35         if (compressor.set_output_file(argv[5]) == ERROR) {
36             std::cerr << "El archivo no se pudo crear" << '\n';
37             return ERROR;
38         }
39     }
40
41     compressor.compress();
42     return SUCCESS;
43 }

```

oct 01, 19 23:38

CompressorThread.h

Page 1/1

```

1  #ifndef _COMPRESSOR_THREAD_H_
2  #define _COMPRESSOR_THREAD_H_
3
4  #include "ProtectedQueue.h"
5  #include "BlockBuffer.h"
6  #include "Reader.h"
7  #include <stdint>
8  #include <thread>
9
10 class CompressorThread {
11     BlockBuffer buffer;
12     int curr_block;
13     int off_blocks;
14
15     Reader& reader;
16
17     ProtectedQueue& queue;
18     std::thread thread;
19
20 private:
21     void compress();
22
23 public:
24     CompressorThread(size_t block_len,
25                     int start,
26                     int off_block,
27                     Reader& reader,
28                     ProtectedQueue& queue);
29
30     void run();
31
32     void join();
33
34     ~CompressorThread();
35 };
36
37 #endif

```

oct 01, 19 23:38

CompressorThread.cpp

Page 1/1

```

1  #include "CompressorThread.h"
2  #include "ProtectedQueue.h"
3  #include "BlockBuffer.h"
4  #include "Block.h"
5  #include "Reader.h"
6  #include <iostream>
7  #include <cstring>
8  #include <stdint>
9  #include <mutex>
10
11 #define DW_BYTES 4
12 #define SUCCESS 0
13 #define ERROR 1
14
15 /*-----Public-----*/
16 CompressorThread::CompressorThread(size_t block_len,
17                                   int start,
18                                   int off_blocks,
19                                   Reader& reader,
20                                   ProtectedQueue& queue):
21     buffer(block_len),
22     reader(reader),
23     queue(queue)
24 {
25     this->off_blocks = off_blocks;
26     this->curr_block = start;
27 }
28
29 void CompressorThread::run() {
30     this->thread = std::thread(&CompressorThread::compress, this);
31 }
32
33 void CompressorThread::join() {
34     this->thread.join();
35 }
36
37 /*-----Private-----*/
38 void CompressorThread::compress() {
39     while (this->reader.set_and_read_block(this->curr_block,
40                                         this->buffer) > 0) {
41         Block* block = this->buffer.create_compressed_block();
42         this->queue.push(block);
43         this->curr_block = this->curr_block + this->off_blocks;
44     }
45     this->queue.close();
46 }
47
48 CompressorThread::~CompressorThread() {
49     //Dont do anything
50 }

```

oct 01, 19 23:38

Compressor.h

Page 1/1

```

1  #ifndef _COMPRESSOR_H_
2  #define _COMPRESSOR_H_
3
4  #include "CompressorThread.h"
5  #include "WriterThread.h"
6  #include "Reader.h"
7  #include "Writer.h"
8  #include <vector>
9  #include <mutex>
10 #include <iostream>
11
12 class Compressor {
13     Reader reader;
14     Writer writer;
15
16     WriterThread* wtr_thread;
17     std::vector<ProtectedQueue> queues;
18     std::vector<CompressorThread*> cmp_threads;
19
20 private:
21     void init_queues(size_t max_q_len, int num_thrds);
22     void init_threads(size_t block_len, int num_thrds);
23
24 public:
25     explicit Compressor(int num_thrds, size_t max_q_len, size_t block_len);
26
27     int set_input_file(const char* i_filename);
28
29     int set_output_file(const char* o_filename);
30
31     void compress();
32
33     ~Compressor();
34 };
35
36 #endif

```

oct 01, 19 23:38

Compressor.cpp

Page 1/1

```

1  #include "Compressor.h"
2  #include "Reader.h"
3  #include "Writer.h"
4  #include "ProtectedQueue.h"
5  #include "CompressorThread.h"
6  #include <iostream>
7
8  /*-----Public-----*/
9  Compressor::Compressor(int num_thrds, size_t max_q_len, size_t block_len):
10     reader(block_len)
11 {
12     this->init_queues(max_q_len, num_thrds);
13     this->init_threads(block_len, num_thrds);
14     this->wtr_thread = new WriterThread(this->queues, this->writer);
15 }
16
17 int Compressor::set_input_file(const char* i_filename) {
18     return this->reader.set_file(i_filename);
19 }
20
21 int Compressor::set_output_file(const char* o_filename) {
22     return this->writer.set_file(o_filename);
23 }
24
25 void Compressor::compress() {
26     for (size_t i = 0; i < cmp_threads.size(); i++) {
27         this->cmp_threads[i]->run();
28     }
29     this->wtr_thread->run();
30     for (size_t i = 0; i < cmp_threads.size(); i++) {
31         this->cmp_threads[i]->join();
32     }
33     this->wtr_thread->join();
34 }
35
36 /*-----Private-----*/
37 void Compressor::init_threads(size_t block_len, int num_thrds) {
38     for (int i = 0; i < num_thrds; i++) {
39         CompressorThread* cmp_thread = new CompressorThread(block_len,
40                                                                i,
41                                                                num_thrds,
42                                                                this->reader,
43                                                                this->queues[i]);
44         this->cmp_threads.push_back(cmp_thread);
45     }
46 }
47
48 void Compressor::init_queues(size_t max_q_len, int num_thrds) {
49     for (int i = 0; i < num_thrds; i++) {
50         this->queues.emplace_back(ProtectedQueue(max_q_len));
51     }
52 }
53
54 Compressor::~Compressor() {
55     delete this->wtr_thread;
56     for (size_t i = 0; i < this->cmp_threads.size(); i++) {
57         delete this->cmp_threads[i];
58     }
59 }

```

oct 01, 19 23:38

Block.h

Page 1/1

```

1  #ifndef _BLOCK_H_
2  #define _BLOCK_H_
3
4  #include "Writer.h"
5  #include <stdint>
6  #include <bitset>
7  #include <vector>
8  #include <fstream>
9  #include "Bitset.h"
10
11 class Block {
12     Bitset bitset;
13     uint8_t bits;
14     uint32_t ref;
15
16     std::vector<uint32_t> numbers;
17
18     private:
19         uint32_t find_max();
20
21         uint32_t find_min();
22
23         void add_padding();
24
25         void numbers_to_bits();
26
27         void subtract_reference();
28
29         uint8_t get_bits(uint32_t number);
30
31     public:
32         explicit Block(const std::vector<uint32_t> numbers);
33
34         void print_in_file(Writer& writer);
35
36         void compress();
37
38         ~Block();
39 };
40
41 #endif

```

oct 01, 19 23:38

Block.cpp

Page 1/2

```

1  #include "Block.h"
2  #include "Writer.h"
3  #include "Bitset.h"
4  #include <stdint>
5  #include <iostream>
6  #include <cmath>
7  #include <vector>
8  #include <bitset>
9
10
11 /*-----Public-----*/
12 Block::Block(const std::vector<uint32_t> numbers) {
13     this->numbers.assign(numbers.begin(), numbers.end());
14     this->ref = this->bits = 0;
15 }
16
17 void Block::print_in_file(Writer& writer) {
18     writer.write_reference(this->ref);
19     writer.write_bits(this->bits);
20     this->bitset.print_in_file(writer);
21 }
22
23 void Block::compress() {
24     this->ref = this->find_min();
25     this->subtract_reference();
26
27     uint32_t max = this->find_max();
28     this->bits = this->get_bits(max);
29
30     this->bitset.set_size(this->bits * this->numbers.size());
31
32     this->numbers_to_bits();
33 }
34
35 /*-----Private-----*/
36 uint32_t Block::find_min() {
37     uint32_t curr_min = this->numbers[0];
38     for (size_t i = 1; i < this->numbers.size(); i++) {
39         if (curr_min > this->numbers[i]) {
40             curr_min = this->numbers[i];
41         }
42     }
43     return curr_min;
44 }
45
46 uint32_t Block::find_max() {
47     uint32_t curr_max = this->numbers[0];
48     for (size_t i = 1; i < this->numbers.size(); i++) {
49         if (curr_max < this->numbers[i]) {
50             curr_max = this->numbers[i];
51         }
52     }
53     return curr_max;
54 }
55
56 void Block::subtract_reference() {
57     for (size_t i = 0; i < this->numbers.size(); i++) {
58         this->numbers[i] = this->numbers[i] - this->ref;
59     }
60 }
61
62 uint8_t Block::get_bits(uint32_t number) {
63     if (number == 0) {
64         return 0;
65     }
66     return (uint8_t)log2(number) + 1;

```


oct 01, 19 23:38

Block.cpp

Page 2/2

```

67 }
68
69 void Block::numbers_to_bits() {
70     std::bitset<32> bitset;
71
72     for (size_t i = 0; i < this->numbers.size(); i++) {
73         bitset = this->numbers[i];
74         for (int j = 0; j < this->bits; j++) {
75             this->bitset.push_bit(bitset[this->bits - 1 - j]);
76         }
77     }
78 }
79
80 Block::~Block() {}

```

oct 01, 19 23:38

BlockBuffer.h

Page 1/1

```

1  #ifndef _BLOCK_BUFFER_H_
2  #define _BLOCK_BUFFER_H_
3
4  #include "Block.h"
5  #include <stdint>
6  #include <vector>
7  #include <stdbool>
8  #include <iostream>
9
10 #define DW_BYTES 4
11
12 class BlockBuffer {
13     size_t block_len;
14     std::vector<uint32_t> buffer;
15
16     private:
17         void complete_buffer();
18
19     public:
20         explicit BlockBuffer(size_t block_len);
21
22         bool is_full();
23
24         int numbers_stored();
25
26         Block* create_compressed_block();
27
28         void add_number(const char* str_number);
29
30         ~BlockBuffer();
31 };
32
33 #endif

```

oct 01, 19 23:38

BlockBuffer.cpp

Page 1/1

```

1  #include "BlockBuffer.h"
2  #include "Block.h"
3  #include <stdint>
4  #include <stdbool>
5  #include <iostream>
6  #include <string>
7  #include <arpa/inet.h>
8
9  /*-----Public-----*/
10 BlockBuffer::BlockBuffer(size_t block_len) {
11     this->block_len = block_len;
12 }
13
14 bool BlockBuffer::is_full() {
15     return this->buffer.size() >= this->block_len;
16 }
17
18 int BlockBuffer::numbers_stored() {
19     return this->buffer.size();
20 }
21
22 void BlockBuffer::add_number(const char* str_number) {
23     uint32_t number;
24     memcpy(&number, str_number, 4);
25     this->buffer.push_back(ntohl(number));
26 }
27
28 Block* BlockBuffer::create_compressed_block() {
29     if (this->buffer.size() > 1) {
30         //To avoid zero brick case
31         this->complete_buffer();
32     }
33     Block* block = new Block(buffer);
34     this->buffer.clear();
35     block->compress();
36     return block;
37 }
38
39 /*-----Private-----*/
40 void BlockBuffer::complete_buffer() {
41     if (this->buffer.size() < this->block_len) {
42         int last_pos = this->buffer.size() - 1;
43         while (this->buffer.size() < this->block_len) {
44             this->buffer.push_back(this->buffer[last_pos]);
45         }
46     }
47 }
48
49 BlockBuffer::~BlockBuffer() {}

```

oct 01, 19 23:38

Bitset.h

Page 1/1

```

1  #ifndef _BITSET_H_
2  #define _BITSET_H_
3
4  #include "Writer.h"
5  #include <vector>
6  #include <stdint>
7  #include <fstream>
8  #include <stdbool>
9
10 class Bitset {
11     int size;
12     int curr_pos;
13     std::vector<bool> bits;
14
15     private:
16         void complete_number();
17         void calculate_padding();
18         uint8_t get_byte_to_print(char* number_by_bit);
19
20     public:
21         Bitset();
22         void push_bit(bool bit);
23         void set_size(int size);
24         void print_in_file(Writer& writer);
25 };
26
27 #endif

```

oct 01, 19 23:38

Bitset.cpp

Page 1/1

```

1  #include "Bitset.h"
2  #include "Writer.h"
3  #include <cstring>
4  #include <stdint>
5  #include <iostream>
6  #include <cstdbool>
7
8  #define BINARY +2
9
10
11 /*-----Public-----*/
12 Bitset::Bitset() {
13     this->size = 0;
14     this->curr_pos = 0;
15 }
16
17 void Bitset::set_size(int size) {
18     this->size = size;
19     this->calculate_padding();
20 }
21
22 void Bitset::push_bit(bool bit) {
23     this->bits.push_back(bit);
24     this->curr_pos++;
25 }
26
27 void Bitset::print_in_file(Writer& writer) {
28     this->complete_number();
29     char number_by_bit[8];
30
31     int offset = 0;
32     for (int i = 0; i < this->size; i++) {
33         number_by_bit[i - offset] = this->bits[i]?'1':'0';
34         if ((i+1) % 8 == 0) {
35             writer.write_number(number_by_bit);
36             offset = offset + 8;
37         }
38     }
39 }
40
41 /*-----Private-----*/
42 void Bitset::calculate_padding() {
43     while (this->size % 8 != 0) {
44         this->size++;
45     }
46 }
47
48 void Bitset::complete_number() {
49     for (int i = this->curr_pos; i < this->size; i++) {
50         this->bits[i] = false;
51     }
52 }

```

oct 01, 19 23:38

Table of Content

Page 1/1

1	Table of Contents				
2	1 WriterThread.h..... sheets	1 to	1 (1) pages	1- 1	34 lines
3	2 WriterThread.cpp.... sheets	1 to	1 (1) pages	2- 2	58 lines
4	3 Writer.h..... sheets	2 to	2 (1) pages	3- 3	29 lines
5	4 Writer.cpp..... sheets	2 to	2 (1) pages	4- 4	49 lines
6	5 Reader.h..... sheets	3 to	3 (1) pages	5- 5	32 lines
7	6 Reader.cpp..... sheets	3 to	3 (1) pages	6- 6	63 lines
8	7 ProtectedQueue.h.... sheets	4 to	4 (1) pages	7- 7	36 lines
9	8 ProtectedQueue.cpp.. sheets	4 to	5 (2) pages	8- 9	70 lines
10	9 main.cpp..... sheets	5 to	5 (1) pages	10- 10	44 lines
11	10 CompressorThread.h.. sheets	6 to	6 (1) pages	11- 11	38 lines
12	11 CompressorThread.cpp sheets	6 to	6 (1) pages	12- 12	51 lines
13	12 Compressor.h..... sheets	7 to	7 (1) pages	13- 13	38 lines
14	13 Compressor.cpp..... sheets	7 to	7 (1) pages	14- 14	60 lines
15	14 Block.h..... sheets	8 to	8 (1) pages	15- 15	42 lines
16	15 Block.cpp..... sheets	8 to	9 (2) pages	16- 17	81 lines
17	16 BlockBuffer.h..... sheets	9 to	9 (1) pages	18- 18	34 lines
18	17 BlockBuffer.cpp..... sheets	10 to	10 (1) pages	19- 19	50 lines
19	18 Bitset.h..... sheets	10 to	10 (1) pages	20- 20	33 lines
20	19 Bitset.cpp..... sheets	11 to	11 (1) pages	21- 21	53 lines