**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

|  |  |  |  |
| --- | --- | --- | --- |
|  | **ПРОГРАММНЫЙ МОДУЛЬ ДЛЯ ВЫДЕЛЕНИЯ ЛОГИЧЕСКИ СВЯЗАННЫХ ПОТОКОВ В ВЫСОКОСКОРОСТНОМ СЕТЕВОМ ТРАФИКЕ**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.05.03-01 01-1-ЛУ** | | |
|  |  | |
| Исполнитель:  студент группы БПИ197  \_\_\_\_\_\_\_\_\_\_\_\_ / Глущенко З. С./  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2022 г. | |
|  | | |
| **Москва 2022** | |  |

|  |  |
| --- | --- |
| ***Подп. и дата*** |  |
| ***Инв. № дубл.*** |  |
| ***Взам. инв. №*** |  |
| ***Подп. и дата*** |  |
| ***Инв. № подл*** |  |

Департамент программной инженерии

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО  Преподаватель факультета компьютерных наук, доцент базовой кафедры «Системное программирование» ИСП ран, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. И. Гетьман  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г.  СОГЛАСОВАНО  Руководитель департамента программной инженерии, доцент факультета компьютерных наук, канд. экон. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. А. Лебедев  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»,  канд. техн .наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. |

|  |  |  |  |
| --- | --- | --- | --- |
| УТВЕРЖДЕНО  RU.17701729.05.03-01 81 01-1-ЛУ | |  |  |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **ПРОГРАММНЫЙ МОДУЛЬ ДЛЯ ВЫДЕЛЕНИЯ ЛОГИЧЕСКИ СВЯЗАННЫХ ПОТОКОВ В ВЫСОКОСКОРОСТНОМ СЕТЕВОМ ТРАФИКЕ**  **Текст программы**  **RU.17701729.05.03-01 81 01-1**  **Листов 20** | | | | |
|  |  | | | |
|  | | | |
| **Москва 2022** | | | | |
|  |  | | | | |

|  |  |
| --- | --- |
|  |  |
|  |  |

СОДЕРЖАНИЕ

[**1.** **Назначение программы** 4](#_Toc100920725)

[**1.1.** **Директория ftp\_extractor** 4](#_Toc100920726)

[**1.1.1.** **Код файла ftp\_** **ftp\_extractor.h** 4](#_Toc100920727)

[**1.1.2.** **Код файла ftp\_extractor.cpp** 6](#_Toc100920728)

[**1.1.3.** **Код файла main.cpp** 10](#_Toc100920729)

[**1.2.** **Директория stream\_merger** 10](#_Toc100920730)

[**1.2.1.** **Код файла** **StreamMerger.h** 10](#_Toc100920731)

[**1.2.2.** **Код файла StreamMerger.cpp** 13](#_Toc100920732)

[**1.2.3.** **Код файла main.cpp** 19](#_Toc100920733)

[**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ** 20](#_Toc100920734)

1. **Назначение программы**
   1. **Директория ftp\_extractor**
      1. **Код файла ftp\_** **ftp\_extractor.h**

#ifndef FTP\_EXTRACTOR\_H

#define FTP\_EXTRACTOR\_H

#include "common/ProcessModule.h"

namespace Pr {

/\*\*

\* @brief The FTP extractor is a class for extracting information from packets

\* and sending it to the appropriate modules.

\*/

class ftp\_extractor : public ProcessModule, public MessageProcessor<ftp\_extractor> {

public:

ftp\_extractor();

virtual ~ftp\_extractor();

/\*\*

\* @brief declareResourcesUsage - function to declare queues, statistics and other module resources

\* @return true if all declarations are successful

\*/

virtual bool declareResourcesUsage();

/\*\*

\* @brief acquireResources - function to get parameters from config and connect queues

\* @return true in case of success

\*/

virtual bool acquireResources();

protected:

/\*\*

\* @brief processMessage - function to process a single message from the input queue

\* @param msg - pointer to the message in the queue of general type QueueMessage

\* @param rec - pointer to the PacketRecord in the packet buffer

\* @return zero in the end of the message processing (here we chose not to return any error codes)

\*/

int processMessage(QueueMessage\* msg, PacketRecord\* rec);

/\*\*

\* @brief sendSignal - function to send a signal to the output queue; will send until the success of the operation

\* @param output\_queue - pointer to the output queue of type T

\* @param signal - pointer to signal to send of type T

\* @return

\*/

template<typename T>

void sendSignal(MessageQueue<T>\* output\_queue, T signal);

/\*\*

\* @brief parseFTP - function to extract an address from control connection packet

\* @param packMess - pointer to the message in the queue of type PacketMessage

\* @param rec - pointer to the PacketRecord in the packet buffer

\* @return pairSignal - reference to the pair signal variable

\* @return shift - offset depending on connection mode

\*/

void parseFTP(PacketMessage\* packMess, PacketRecord\* rec, PairStreamsSignalV4& pairSignal, int shift);

// queues

/\*\*

\* @brief m\_packets - input packets queue

\*/

MessageQueue<StreamPacket>\* m\_packets;

/\*\*

\* @brief m\_input\_queue - The queue of output pair signals

\*/

MessageQueue<PairStreamsSignalV4>\* m\_output\_merger;

/\*\*

\* @brief m\_input\_queue - The queue of output pair signals

\*/

MessageQueue<PairStreamsSignalV4>\* m\_output\_converter;

/\*\*

\* @brief m\_input\_queue - The queue of output SelectSignalV4 signals

\*/

MessageQueue<SelectSignalV4>\* m\_output\_selector;

// module statistics;

/\*\*

\* @brief m\_lostPackets - counter of lost packets

\*/

uint64\_t \*m\_lostPackets;

/\*\*

\* @brief m\_mutex - mutex to work with inout packets

\*/

boost::mutex m\_mutex;

};

}

#endif

* + 1. **Код файла ftp\_extractor.cpp**

#include "ftp\_extractor.h"

// config parameters

// parameter with '?' is optional

const char \* spec = R"SPEC(

{

"packetQueue" : "string",

"packetBuffer" : "string",

"pairSignals\_merger" : "string",

"SelectSignals\_selector" : "string",

"pairSignals\_converter" : "string"

}

)SPEC";

Pr::ftp\_extractor::ftp\_extractor() : ProcessModule("ftp\_extractor", spec),

MessageProcessor<ftp\_extractor>(const\_cast<ftp\_extractor\*>(this)) {

m\_packets = nullptr;

m\_output\_merger = nullptr;

m\_output\_converter = nullptr;

m\_output\_selector = nullptr;

}

Pr::ftp\_extractor::~ftp\_extractor() {

// flush the output queue

m\_output\_merger->flush();

m\_output\_converter->flush();

m\_output\_selector->flush();

m\_packets = nullptr;

m\_output\_merger = nullptr;

m\_output\_converter = nullptr;

m\_output\_selector = nullptr;

}

// here we create queues and statistical parameters to watch

bool Pr::ftp\_extractor::declareResourcesUsage() {

// here we declare the input queue of type PacketMessage

// names of the queue and the packet buffer are derived from the config file

// it also automatically starts counting some statistics for this queue

declareQueueUsage(m\_config.value<std::string>("packetQueue"),

{"PacketMessage", true, false, false, -1,

"Packets received", "Packets lost",

"Avg. processing time", "Max throughput",

(MessageQueueInterface\*\*)&m\_packets,

m\_config.value<std::string>("packetBuffer"), false});

// declare queues

declareQueueUsage(m\_config.value<std::string>("pairSignals\_merger"), "PairStreamsSignalV4", false, true, (MessageQueueInterface\*\*)&m\_output\_merger);

declareQueueUsage(m\_config.value<std::string>("SelectSignals\_selector"), "SelectSignalV4", false, true, (MessageQueueInterface\*\*)&m\_output\_selector);

declareQueueUsage(m\_config.value<std::string>("pairSignals\_converter"), "PairStreamsSignalV4", false, true, (MessageQueueInterface\*\*)&m\_output\_converter);

return true;

}

bool Pr::ftp\_extractor::acquireResources()

{

m\_lostPackets = acquirePointerToStatParam("Packets lost");

// set input queue listener

if(!setQueueListener(m\_packets, this)) {

return false;

}

return true;

}

// this is the function that is called each time we receive a new packet in the input queue

// it is the main processing function

int Pr::ftp\_extractor::processMessage(QueueMessage\* msg, PacketRecord\* rec)

{

PacketMessage\* mess = (PacketMessage\*)msg;

// we use the mutex to separate the processing of each packet

boost::lock\_guard<boost::mutex> guard(m\_mutex);

IpHeader\* header = (IpHeader\*)rec->payload();

if (header->proto != TCP)

{

return 0;

}

TcpHeader\* tcp = (TcpHeader\*)(header->payload());

int ftp\_payload = header->headerLength() + tcp->headerLength();

if (ftp\_payload + 4 < rec->dataSize())

{

// interpret first 4 ftp\_payload's chars as hex digit

const int32\_t& hx= \*reinterpret\_cast<int32\_t\*>(rec->payload() + ftp\_payload);

const int32\_t hex\_port = 0x54524f50;

const int32\_t hex\_pasv = 0x20373232;

const int port\_shift = ftp\_payload + 5;

const int pasv\_shift = ftp\_payload + 27;

PairStreamsSignalV4 pairSignal;

if(hx == hex\_port)

{

Logger::log(Logger::Info, "Identify PORT");

parseFTP(mess, rec, pairSignal, port\_shift);

pairSignal.childAddr.srcPort = htons(20);

SelectSignalV4 sv4;

sv4.addr = pairSignal.childAddr;

sv4.flags = pairSignal.flags;

sendSignal(m\_output\_merger, pairSignal);

sendSignal(m\_output\_selector, sv4);

}

else if (hx == hex\_pasv)

{

Logger::log(Logger::Info, "Identify PASV 227");

parseFTP(mess, rec, pairSignal, pasv\_shift);

sendSignal(m\_output\_converter, pairSignal);

}

else

{

Logger::log(Logger::Error, "recieved wrong packet!");

(\*m\_lostPackets)++;

}

}

m\_output\_merger->flush();

m\_output\_converter->flush();

m\_output\_selector->flush();

return 0;

}

template<typename T>

void Pr::ftp\_extractor::sendSignal(MessageQueue<T>\* output\_queue, T signal)

{

// try sending until we reach the success status

while(output\_queue->send(signal) != Infra::MessageQueueInterface::Sent) {}

output\_queue->flush();

}

void Pr::ftp\_extractor::parseFTP(PacketMessage\* packMess, PacketRecord\* rec, PairStreamsSignalV4& pairSignal, int shift)

{

IpHeader\* header = (IpHeader\*)rec->payload();

int32\_t tmpNum = 0;

int32\_t ip = 0;

int16\_t port = 0;

int curIteration = 0; // 4 - Ip bytes, 6 - port bytes

// bool error = true;

for (\_\_uint32\_t i = shift; i < rec->dataSize(); ++i)

{

char curSymbol = \*(rec->payload() + i);

// если это цифра

if(curSymbol >= '0' && curSymbol <= '9')

{

tmpNum = tmpNum \* 10 + (curSymbol - '0');

}

else

{

// сдвигаем на 8 бит исходное и выгружаем tmpNum

++curIteration;

if(curIteration <= 4)

{

ip = (ip << 8) + tmpNum;

tmpNum=0;

}

else if(curIteration <= 6)

{

port = (port<< 8) + tmpNum;

tmpNum=0;

}

else

{

// если дошли до сюда, значит удачно считали ip & port

// error = false;

break;

}

}

}

pairSignal.creationTime = rec->timestamp();

pairSignal.flags = 1u;

// parent - info from recieved packet

//pairSignal.parentId = mess->streamId;

rec->toAddrInfo(pairSignal.parentAddr);

// child - parsed info

pairSignal.childAddr.proto = TCP;

pairSignal.childAddr.destIp = ip;

pairSignal.childAddr.destPort = port;

pairSignal.childAddr.srcIp = header->destIp;

//pairSignal.childAddr.srcPort = 0;

pairSignal.childAddr.destIp = htonl(pairSignal.childAddr.destIp);

pairSignal.childAddr.destPort = htons(pairSignal.childAddr.destPort);

// Это packet\_filter -> нужно подать сигналы на отбор

// This is control connection!

SelectSignalV4 selector;

selector.addr = pairSignal.parentAddr;

selector.creationTime = pairSignal.creationTime;

selector.flags = pairSignal.flags;

sendSignal(m\_output\_selector, selector);

}

* + 1. **Код файла main.cpp**

#include "ftp\_extractor.h"

int main(int argc, char\* argv[])

{

return Pr::ftp\_extractor().run(argc, argv);

}

* 1. **Директория stream\_merger**
     1. **Код файла** **StreamMerger.h**

#ifndef STREAMMERGER\_H

#define STREAMMERGER\_H

#include <pcap.h>

#include <unordered\_map>

#include <common/ProcessModule.h>

#include <infra/includes/OldestFirstPool.h>

#include <common/StreamTypes.h>

namespace Pr

{

/\*\*

\* @brief The StreamMerger is a class for merging packets of several connected flows into one pcap file

\*/

class StreamMerger : public ProcessModule, public MessageProcessor<Pr::StreamMerger>

{

struct PairStreamNode

{

StreamDescrV4 parent;

uint32\_t childCount;

uint32\_t recievedChildCount = 0;

bool isVisited = false;

Dequeue children;

};

struct StreamNode : StreamDescrV4, DequeueNode

{

PairStreamNode\* parent;

};

public:

/\*\*

\* @brief PacketManager - simple constructor

\*/

StreamMerger();

~StreamMerger();

virtual bool declareResourcesUsage();

virtual bool acquireResources();

private:

void removeStream(StreamDescrV4\* stream);

void mergeStreams(PairStreamNode\* parent);

int processMessage(QueueMessage\* msg, PacketRecord\* rec);

int processSignal(QueueMessage\* msg, PacketRecord\* rec);

void startMerge(PairStreamNode\* parent);

// resources

/\*\*

\* @brief m\_addr2parent - map from stream addrs to parent streams

\*/

std::unordered\_map<AddrInfoV4, PairStreamNode\*> m\_addr2parent;

/\*\*

\* @brief m\_childStreams - pool for child streams

\*/

OldestFirstPool<StreamNode> m\_childStreams;

/\*\*

\* @brief m\_parentStreams - pool for parent streams

\*/

OldestFirstPool<PairStreamNode> m\_parentStreams;

/\*\*

\* @brief m\_streamIn - input queue for stream packets

\*/

MessageQueue<SelectSignalV4>\* m\_streamIn;

/\*\*

\* @brief m\_streamIn - input queue for stream packets

\*/

MessageQueue<PairStreamsSignalV4>\* m\_pairsIn;

/\*\*

\* @brief m\_path - path for pcap files

\*/

std::string m\_pathFrom;

/\*\*

\* @brief m\_path - path for pcap files

\*/

std::string m\_pathTo;

// module statistics

/\*\*

\* @brief m\_maxStreamsCount - max streams count

\*/

uint32\_t m\_maxStreamsCount;

/\*\*

\* @brief m\_lostPackets - pair merged count

\*/

uint64\_t\* m\_pairsMerged;

/\*\*

\* @brief m\_notPairedStreams - streams with no pair

\*/

uint64\_t\* m\_notPairedStreams;

/\*\*

\* @brief m\_maxChilds - maximum children count with one parent

\*/

uint64\_t\* m\_maxChilds;

/\*\*

\* @brief m\_lostParents - lost parent streams count

\*/

uint64\_t\* m\_lostParents;

/\*\*

\* @brief m\_lostChildren - lost child streams count

\*/

uint64\_t\* m\_lostChildren;

/\*\*

\* @brief m\_mergeResults - merge results count

\*/

uint64\_t\* m\_mergeResults;

/\*\*

\* @brief m\_mutex - mutex for change paired streams

\*/

boost::mutex m\_mutex;

};

}

#endif //STREAMMERGER\_H

* + 1. **Код файла StreamMerger.cpp**

#include "StreamMerger.h"

#include <cstdint>

#include <cstdio>

#include <boost/format.hpp>

const char \* spec = R"SPEC(

{

"maxStreamsCount" : "int",

"inputPath" : "string",

"outputPath" : "string",

"streamsIn" : "string",

"pairSignalsIn" : "string"

}

)SPEC";

Pr::StreamMerger::StreamMerger() : ProcessModule("StreamMerger", spec),

MessageProcessor<StreamMerger>(const\_cast<StreamMerger\*>(this))

{

}

Pr::StreamMerger::~StreamMerger()

{

m\_childStreams.deinit();

m\_parentStreams.deinit();

}

bool Pr::StreamMerger::declareResourcesUsage()

{

m\_maxStreamsCount = (uint32\_t)m\_config.value<int>("maxStreamsCount");

// m\_pathFrom = dataPath(m\_config.value<std::string>("inputPath"));

// m\_pathTo = outputFile(m\_config.value<std::string>("outputPath"));

m\_pathFrom = m\_config.value<std::string>("inputPath");

m\_pathTo = m\_config.value<std::string>("outputPath");

declareStatParam("Pairs merged", &m\_pairsMerged);

declareStatParam("Not paired streams", &m\_notPairedStreams);

declareStatParam("Maximum children", &m\_maxChilds);

declareStatParam("Lost parents", &m\_lostParents);

declareStatParam("Lost children", &m\_lostChildren);

declareStatParam("Merge results", &m\_mergeResults);

declareQueueUsage(m\_config.value<std::string>("streamsIn"), {"SelectSignalV4", true, false, false, -1, "Streams received", "", "", "", (MessageQueueInterface\*\*)&m\_streamIn});

declareQueueUsage(m\_config.value<std::string>("pairSignalsIn"), {"PairStreamsSignalV4", true, false, false, -1, "Signals received", "", "", "", (MessageQueueInterface\*\*)&m\_pairsIn});

return true;

}

bool Pr::StreamMerger::acquireResources()

{

if(!m\_parentStreams.init(m\_maxStreamsCount))

{

std::string err= boost::str(boost::format("Can't create pool for parent streams with size %1%") % m\_maxStreamsCount);

Infra::Logger::log(Logger::Error, err);

return false;

}

if(!m\_childStreams.init(m\_maxStreamsCount))

{

std::string err= boost::str(boost::format("Can't create pool for child streams with size %1%") % m\_maxStreamsCount);

Infra::Logger::log(Logger::Error, err);

return false;

}

if(!setQueueListener(m\_streamIn, this))

{

return false;

}

if(!setQueueListener(m\_pairsIn, this, function(&Pr::StreamMerger::processSignal)))

{

return false;

}

return true;

}

void Pr::StreamMerger::removeStream(StreamDescrV4\* stream)

{

std::unordered\_map<AddrInfoV4, PairStreamNode\*>::iterator iter;

iter = m\_addr2parent.find(stream->addr);

if(iter != m\_addr2parent.end())

{

m\_addr2parent.erase(iter);

}

std::string fileName = boost::str(boost::format("%1%%2%") % m\_pathFrom % stream->toString());

if(remove(fileName.c\_str()) != 0)

{

std::string err= boost::str(boost::format("Can't remove stream file %1%") % fileName);

Infra::Logger::log(Logger::Error, err);

}

}

void Pr::StreamMerger::mergeStreams(PairStreamNode\* parent)

{

DequeueNode\* cur = parent->children.front();

if(cur != 0)

{

std::string parentFile = parent->parent.toString();

std::string command = boost::str(boost::format("mergecap -w %1%%2% %3%%4%") % m\_pathTo % parentFile % m\_pathFrom % parentFile);

int count = 0;

while(cur != 0)

{

StreamNode\* node = (StreamNode\*)cur;

command += boost::str(boost::format(" %1%%2%") % m\_pathFrom % node->toString());

count++;

cur = cur->next;

}

std::string info = boost::str(boost::format("Merging (%1%) streams with parent (%2%)") % count % parentFile);

Infra::Logger::log(Logger::Warning, info);

if(system(command.c\_str()) == 0)

{

(\*m\_pairsMerged) += count;

(\*m\_mergeResults)++;

}

else

{

(\*m\_lostParents)++;

(\*m\_lostChildren) += count;

}

info = boost::str(boost::format("Merged"));

Infra::Logger::log(Logger::Warning, info);

}

}

int Pr::StreamMerger::processSignal(QueueMessage\* msg, PacketRecord\* rec)

{

PairStreamsSignalV4\* mess = static\_cast<PairStreamsSignalV4\*>( msg);

boost::lock\_guard<boost::mutex> guard(m\_mutex);

std::unordered\_map<AddrInfoV4, PairStreamNode\*>::iterator iter;

iter = m\_addr2parent.find(mess->parentAddr);

if(iter == m\_addr2parent.end())

{

PairStreamNode\* parent = m\_parentStreams.get();

if(!m\_parentStreams.isEmpty(parent))

{

(\*m\_lostParents)++;

iter = m\_addr2parent.find(parent->parent.addr);

if(iter != m\_addr2parent.end())

{

m\_addr2parent.erase(iter);

}

DequeueNode\* cur = parent->children.front();

while(cur != 0)

{

StreamNode\* node = (StreamNode\*)cur;

removeStream(node);

m\_childStreams.free(node);

(\*m\_lostChildren)++;

cur = cur->next;

}

}

parent->childCount = 0;

parent->children.clear();

parent->parent.addr = mess->parentAddr;

parent->parent.creationTime = 0;

iter = m\_addr2parent.insert(std::pair<AddrInfoV4, PairStreamNode\*>(mess->parentAddr, parent)).first;

}

// Now parent exist

PairStreamNode\* parent = iter->second;

parent->childCount++;

if(parent->childCount > (\*m\_maxChilds))

{

(\*m\_maxChilds) = parent->childCount;

}

std::string info = boost::str(boost::format("Pair (%1%) + (%2%)") % mess->childAddr.toString() % mess->parentAddr.toString());

Infra::Logger::log(Logger::Warning, info);

m\_addr2parent.insert(std::pair<AddrInfoV4, PairStreamNode\*>(mess->childAddr, parent));

return 0;

}

int Pr::StreamMerger::processMessage(QueueMessage\* msg, PacketRecord\* rec)

{

boost::lock\_guard<boost::mutex> guard(m\_mutex);

SelectSignalV4\* mess = static\_cast<SelectSignalV4\*>(msg);

std::unordered\_map<AddrInfoV4, PairStreamNode\*>::iterator iter;

iter = m\_addr2parent.find(mess->addr);

if(iter == m\_addr2parent.end())

{

(\*m\_notPairedStreams)++;

//std::string info = boost::str(boost::format("Received unpaired stream (%1%)") % mess->addr.toString());

//Infra::Logger::log(Logger::Warning, info);

StreamDescrV4 stream;

stream.addr = mess->addr;

stream.creationTime = mess->creationTime;

removeStream(&stream);

return 0;

}

PairStreamNode\* parent = iter->second;

//Received parent

if(parent->parent.addr == mess->addr)

{

parent->parent.creationTime = mess->creationTime;

if (parent->recievedChildCount == parent->childCount)

{

startMerge(parent);

}

else

{

parent->isVisited = true;

}

}

// Recieve child

else

{

StreamNode\* child = m\_childStreams.get();

if(!m\_childStreams.isEmpty(child))

{

removeStream(child);

child->parent->children.stichQueue(child);

(\*m\_lostChildren)++;

}

child->addr = mess->addr;

child->creationTime = mess->creationTime;

parent->recievedChildCount++;

parent->children.pushBack(child);

if((parent->recievedChildCount == parent->childCount) && parent->isVisited)

{

startMerge(parent);

}

}

return 0;

}

void Pr::StreamMerger::startMerge(PairStreamNode\* parent)

{

mergeStreams(parent);

// delete children

DequeueNode\* cur = parent->children.front();

while(cur != 0)

{

StreamNode\* node = (StreamNode\*)cur;

removeStream(node);

m\_childStreams.free(node);

cur = cur->next;

}

// delete parent

removeStream(&(parent->parent));

m\_parentStreams.free(parent);

}

* + 1. **Код файла main.cpp**

#include "StreamMerger.h"

int main(int argc, char\* argv[])

{

Pr::StreamMerger().run(argc, argv);

}

# **ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированх |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |