Приложение Б

(обязательное)

**Листинг исходного кода**

Б.1 Текст программы «src\networking\udp\_receiver.hpp»

#pragma once

#pragma once

#include <boost/asio.hpp>

#include <boost/asio/ip/address.hpp>

#include <boost/asio/ip/udp.hpp>

#include <config/type/udp\_receiver.hpp>

#include <functional>

#include <models/udp\_buffer.hpp>

#include <optional>

#include <vector>

namespace networking::udp\_receiver {

namespace ip = boost::asio::ip;

class UdpReceiver {

public:

using HandlerType = std::function<void(models::udp\_buffer::DataBuffer&)>;

UdpReceiver(const config::udp\_receiver::Configuration& config,

const HandlerType handler)

: started\_(false),

thread\_count\_(config.receive\_threads),

ip\_version\_(config.ip\_version),

buffer\_queue\_(config.buffer\_count, config.datagram\_max\_size),

socket\_(io\_service\_),

receiver\_endpoint\_(boost::asio::ip::address\_v4::any(),

config.target\_port),

expected\_source\_ip\_(config.target\_ip),

handler\_(handler){};

void Start();

class ReceiveFailed : std::runtime\_error {

using std::runtime\_error::runtime\_error;

};

class BufferCorrupted : std::runtime\_error {

using std::runtime\_error::runtime\_error;

};

class BufferSizeLessThanRequired : std::runtime\_error {

using std::runtime\_error::runtime\_error;

};

~UdpReceiver() {

io\_service\_.stop();

socket\_.close();

for (auto& t : receive\_threads\_) {

//End threads and supress exceptions if present;

try {

t.join();

} catch (const std::exception&) {}

}

}

private:

void WaitReceive();

void Handle(models::udp\_buffer::DataBuffer& data\_buffer,

ip::udp::endpoint& udp\_source,

const boost::system::error\_code& error, size\_t bytes\_transferred);

bool started\_;

int thread\_count\_;

ip::udp ip\_version\_;

models::udp\_buffer::BufferQueue buffer\_queue\_;

boost::asio::io\_service io\_service\_;

ip::udp::socket socket\_{io\_service\_};

boost::asio::io\_service::work work\_{io\_service\_};

ip::udp::endpoint receiver\_endpoint\_;

std::optional<ip::address> expected\_source\_ip\_;

std::vector<std::thread> receive\_threads\_;

HandlerType handler\_;

};

} // namespace networking::udp\_receiver

Б.2 Текст программы «src\networking\udp\_receiver.cpp»

#include "udp\_receiver.hpp"

#include <fmt/core.h>

#include <boost/bind/bind.hpp>

#include <unordered\_set>

#include <utils/logging.hpp>

namespace networking::udp\_receiver {

namespace {

struct RAIIBufferContainer {

RAIIBufferContainer(

std::unique\_ptr<models::udp\_buffer::DataBuffer> buffer\_to\_return,

models::udp\_buffer::BufferQueue& queue\_to\_return)

: container(std::move(buffer\_to\_return)), queue(queue\_to\_return) {}

~RAIIBufferContainer() { queue.ReleaseBuffer(std::move(container)); }

std::unique\_ptr<models::udp\_buffer::DataBuffer> container;

models::udp\_buffer::BufferQueue& queue;

};

} // namespace

void UdpReceiver::Start() {

if (started\_) {

LOG\_INFO() << "UdpReceiver already started\n";

return;

}

started\_ = true;

socket\_.open(ip\_version\_);

socket\_.bind(receiver\_endpoint\_);

for (int i = 0; i < thread\_count\_; i++) {

receive\_threads\_.push\_back(std::thread([this] {

WaitReceive();

io\_service\_.run();

}));

}

}

void UdpReceiver::WaitReceive() {

auto buffer = buffer\_queue\_.AquireBuffer();

auto endpoint = std::make\_unique<ip::udp::endpoint>();

socket\_.async\_receive\_from(

boost::asio::buffer(buffer->buffer, buffer->buffer.size()), \*endpoint,

[data\_buffer = std::move(buffer), udp\_source = std::move(endpoint), this](

const boost::system::error\_code& error,

size\_t bytes\_transferred) mutable {

if (!data\_buffer) {

throw UdpReceiver::BufferCorrupted("Data buffer ptr is NULL");

}

if (!udp\_source) {

throw UdpReceiver::BufferCorrupted("UdpSource lost");

}

RAIIBufferContainer secured\_buffer(std::move(data\_buffer),

buffer\_queue\_);

Handle(\*secured\_buffer.container, \*udp\_source, error,

bytes\_transferred);

});

}

void UdpReceiver::Handle(models::udp\_buffer::DataBuffer& data\_buffer,

ip::udp::endpoint& udp\_source,

const boost::system::error\_code& error,

size\_t bytes\_transferred) {

static const std::unordered\_set<int> ignored\_errors{

boost::asio::error::operation\_aborted};

if (error) {

if (!ignored\_errors.contains(error.value())) {

LOG\_WARNING() << fmt::format("Receive failed: {} | Val: {} | Cat: {}\n",

error.message(), error.value(),

error.category().name());

}

return;

}

if (expected\_source\_ip\_) {

if (expected\_source\_ip\_ != udp\_source.address()) {

LOG\_INFO() << "Receive blocked for ip: " << udp\_source.address() << "\n";

return;

}

}

if (data\_buffer.buffer.size() < bytes\_transferred) {

throw UdpReceiver::BufferSizeLessThanRequired("Data buffer ptr is NULL");

}

data\_buffer.last\_datagram\_size = bytes\_transferred;

try {

handler\_(data\_buffer);

} catch (std::runtime\_error& e) {

LOG\_INFO() << "Caught untyped exception: " << e.what() << "\n";

}

}

} // namespace networking::udp\_receiver

Б.3 Текст программы «src\networking\udp\_sender.hpp»

#pragma once

#include <boost/asio.hpp>

#include <boost/asio/ip/address.hpp>

#include <boost/asio/ip/udp.hpp>

#include <config/type/udp\_sender.hpp>

#include <vector>

namespace networking::udp\_sender {

namespace ip = boost::asio::ip;

class UdpSender {

public:

struct Package {

std::vector<char> data;

int retries;

};

class DataIncorrectState : std::runtime\_error {

using std::runtime\_error::runtime\_error;

};

UdpSender(const config::udp\_sender::Configuration& config)

: receiver\_endpoint\_(config.target\_ip, config.target\_port),

retry\_count\_(config.retry\_count) {

socket\_.open(config.ip\_version);

}

void Start();

void Send(std::unique\_ptr<Package> data\_to\_send);

~UdpSender() {

io\_service\_.stop();

socket\_.close();

try {

sender\_thread\_->join();

} catch (const std::exception&) {

}

}

private:

ip::udp::endpoint receiver\_endpoint\_;

boost::asio::io\_service io\_service\_;

boost::asio::io\_service::work work\_{io\_service\_};

ip::udp::socket socket\_{io\_service\_};

std::unique\_ptr<std::thread> sender\_thread\_;

int retry\_count\_;

};

} // namespace networking::udp\_sender

Б.4 Текст программы «src\networking\udp\_sender.cpp»

#include "udp\_sender.hpp"

#include <fmt/core.h>

#include <utils/logging.hpp>

namespace networking::udp\_sender {

void UdpSender::Start() {

sender\_thread\_ =

std::make\_unique<std::thread>(std::thread([this] { io\_service\_.run(); }));

}

void UdpSender::Send(std::unique\_ptr<UdpSender::Package> data\_to\_send) {

if (!data\_to\_send) {

throw DataIncorrectState("Data package handles nothing");

}

socket\_.async\_send\_to(

boost::asio::buffer(data\_to\_send->data), receiver\_endpoint\_,

[data\_buffer = std::move(data\_to\_send), this](

const boost::system::error\_code& error,

size\_t bytes\_transferred) mutable {

if (!error && data\_buffer->data.size() == bytes\_transferred) {

/\*Send succeed\*/

return;

}

LOG\_WARNING() << fmt::format(

"Send failed: {} \ndata.size()::{} / actually\_sent::{}\n",

error.what(), data\_buffer->data.size(), bytes\_transferred);

if (data\_buffer->retries > retry\_count\_) {

LOG\_WARNING() << "Data sent failed after retries: "

<< data\_buffer->retries;

return;

}

data\_buffer->retries += 1;

Send(std::move(data\_buffer));

});

}

} // namespace networking::udp\_sender

Б.5 Текст программы «src\modules\fs\_watcher.hpp»

#pragma once

#include <boost/asio.hpp>

#include <config/type/fs\_watcher.hpp>

#include <filesystem>

#include <functional>

#include <optional>

#include <thread>

#include <unordered\_map>

#include <utils/call\_each.hpp>

namespace modules::fs\_watcher {

enum class FileAction { kCreated, kModified, kDeleted };

struct FileDiscription {

std::filesystem::path path;

uintmax\_t size;

std::filesystem::file\_time\_type last\_write\_time;

bool operator==(const FileDiscription& other) const {

return path == other.path && size == other.size &&

last\_write\_time == other.last\_write\_time;

}

};

class FileWatcher {

public:

using FileUpdateHandler =

std::function<void(const FileDiscription&, FileAction)>;

FileWatcher(config::filesystem\_watcher::Configuration& config,

std::optional<FileUpdateHandler> handler)

: folder\_path\_(config.folder\_path), handler\_(handler) {

namespace fs = std::filesystem;

for (auto& file : fs::recursive\_directory\_iterator(folder\_path\_)) {

if (fs::is\_regular\_file(file)) {

cache\_[file.path().string()] =

std::make\_unique<FileDiscription>(FileDiscription{

file.path(), file.file\_size(), file.last\_write\_time()});

}

}

if (handler) {

caller\_ = std::make\_unique<utils::call\_each::CallEach>(

[this]() { UpdateFolder(); }, config.check\_folder\_each);

}

}

void UpdateFolder();

void List(std::function<void(const FileDiscription&)> info\_cb);

private:

using FileWatcherCache =

std::unordered\_map<std::string, std::unique\_ptr<FileDiscription>>;

std::string folder\_path\_;

std::optional<FileUpdateHandler> handler\_;

std::unique\_ptr<utils::call\_each::CallEach> caller\_;

std::mutex mutex\_;

FileWatcherCache cache\_;

};

} // namespace modules::filesystem\_watcher

Б.6 Текст программы «src\modules\fs\_watcher.cpp»

#include "fs\_watcher.hpp"

namespace modules::fs\_watcher {

void FileWatcher::UpdateFolder() {

namespace fs = std::filesystem;

const std::lock\_guard<std::mutex> lock(mutex\_);

FileWatcherCache prev\_cache\_state\_;

cache\_.swap(prev\_cache\_state\_);

for (auto& file :

std::filesystem::recursive\_directory\_iterator(folder\_path\_)) {

if (!fs::is\_regular\_file(file)) {

continue;

}

auto current\_file\_last\_write\_time = std::filesystem::last\_write\_time(file);

auto file\_size = file.file\_size();

auto prev\_file\_state = prev\_cache\_state\_.find(file.path().string());

auto file\_discription =

FileDiscription{file.path(), file\_size, current\_file\_last\_write\_time};

if (prev\_file\_state == prev\_cache\_state\_.end()) {

// New file

if (handler\_) {

handler\_.value()(file\_discription, FileAction::kCreated);

}

cache\_[file.path().string()] =

std::make\_unique<FileDiscription>(std::move(file\_discription));

} else {

// Exists

if (!(\*prev\_file\_state->second == file\_discription)) {

// Has changes

if (handler\_) {

handler\_.value()(file\_discription, FileAction::kModified);

}

}

prev\_cache\_state\_.erase(prev\_file\_state);

}

cache\_[file.path().string()] =

std::make\_unique<FileDiscription>(std::move(file\_discription));

}

for(const auto& [path, description] : prev\_cache\_state\_){

if (handler\_) {

handler\_.value()(\*description, FileAction::kDeleted);

}

}

}

void FileWatcher::List(std::function<void(const FileDiscription&)> info\_cb){

const std::lock\_guard<std::mutex> lock(mutex\_);

for(const auto& [path, description] : cache\_){

info\_cb(\*description);

}

}

} // namespace modules::filesystem\_watcher

Б.7 Текст программы «src\modules\asio\_job\_queue.hpp»

#include <boost/asio.hpp>

#include <boost/bind/bind.hpp>

#include <functional>

#include <iostream>

#include <thread>

namespace modules::asio\_job\_queue {

class AsioJobQueue {

public:

AsioJobQueue(std::size\_t pool\_size) : work\_(io\_service\_) {

for (std::size\_t i = 0; i < pool\_size; ++i) {

threads\_.push\_back(std::thread([this] { io\_service\_.run(); }));

}

}

~AsioJobQueue(){

io\_service\_.stop();

for (auto& t : threads\_) {

//End threads and supress exceptions if present;

try {

t.join();

} catch (const std::exception&) {}

}

}

template <typename Task>

void Shedule(Task task) {

const std::lock\_guard<std::mutex> lock(mutex\_);

io\_service\_.post([this, &task]() { Wrapper(std::function<void()>(task)); });

}

private:

void Wrapper(std::function<void()> task);

boost::asio::io\_service io\_service\_;

boost::asio::io\_service::work work\_;

std::vector<std::thread> threads\_;

std::mutex mutex\_;

};

} // namespace modules::asio\_job\_queue

Б.8 Текст программы «src\modules\asio\_job\_queue.cpp»

#include "asio\_job\_queue.hpp"

namespace modules::asio\_job\_queue {

void AsioJobQueue::Wrapper(std::function<void()> task) {

try {

task();

} catch (const std::exception& e) {

std::cout << "Exception caught in wrapper" << e.what() << "\n";

} catch (const std::runtime\_error& e) {

std::cout << "Exception caught in wrapper" << e.what() << "\n";

}

}

} // namespace modules::asio\_job\_queue

Б.9 Текст программы «src\modules\file\_scheduler.hpp»

#pragma once

#include <fstream>

#include <functional>

#include <models/file\_frame.hpp>

#include <modules/fs\_watcher.hpp>

namespace helpers::file\_scheduler {

class FileReader {

public:

using ReadCallback = std::function<void(models::file\_frame::FileFragment&&)>;

FileReader(const modules::fs\_watcher::FileDiscription& discription,

uint64\_t max\_frame\_size)

: max\_frame\_size\_(max\_frame\_size),

file\_(discription\_.path, std::ios::binary),

discription\_(discription) {}

void Read(ReadCallback callback, models::file\_frame::Action action);

bool OpenOk();

~FileReader() { file\_.close(); }

private:

uint64\_t max\_frame\_size\_;

std::ifstream file\_;

modules::fs\_watcher::FileDiscription discription\_;

};

} // namespace helpers::file\_scheduler

Б.10 Текст программы «src\models\file\_frame.hpp»

#pragma once

#include <filesystem>

#include <string>

#include <vector>

#include <helpers/serialization.hpp>

#include <helpers/to.hpp>

namespace models::file\_frame {

using Payload = std::vector<char>;

enum class Action {

kCreate,

kModify,

kDelete

};

std::string ToString(const Action&);

Action Convert(const std::string&, helpers::to::To<Action>);

struct FileHeader {

std::string path;

uintmax\_t size;

Action action;

std::filesystem::file\_time\_type last\_write\_time;

std::string sha256\_hash;

};

struct FragmentHeader {

uint64\_t part;

uint64\_t parts;

std::string part\_sha256\_hash;

uint64\_t write\_position;

};

struct FileFragment {

FileHeader file\_header;

FragmentHeader fragment\_header;

std::string idempotency\_token;

Payload payload;

};

} // namespace models::file\_frame

namespace helpers::serialization {

SerializedContainer Serialize(

const models::file\_frame::FileFragment& to\_serialize);

models::file\_frame::FileFragment Deserialize(

SerializedContainer::const\_iterator& start\_position,

to::To<models::file\_frame::FileFragment>);

} // namespace helpers::serialization

Б.11 Текст программы «src\models\file\_frame.cpp»

#include "file\_frame.hpp"

#include <unordered\_map>

#include <models/idempotency\_token.hpp>

namespace models::file\_frame {

std::string ToString(const Action& action) {

switch (action) {

case Action::kCreate:

return "create";

case Action::kModify:

return "modify";

case Action::kDelete:

return "delete";

}

}

Action Convert(const std::string& from, helpers::to::To<Action>) {

static const std::unordered\_map<std::string, Action> convert{

{"create", Action::kCreate},

{"modify", Action::kModify},

{"delete", Action::kDelete}};

auto find = convert.find(from);

if (find == convert.end()) {

throw helpers::to::ConvertError(std::string("Can't convert Action from: ") +

from);

}

return find->second;

}

} // namespace models::file\_frame

namespace helpers::serialization {

SerializedContainer Serialize(

const models::file\_frame::FileHeader& to\_serialize) {

SerializedContainer out;

auto write\_time = to\_serialize.last\_write\_time.time\_since\_epoch().count();

auto path\_part = Serialize(to\_serialize.path);

auto size\_part = Serialize(to\_serialize.size);

auto action\_part = Serialize(ToString(to\_serialize.action));

auto last\_write\_part = Serialize(write\_time);

auto sha\_part = Serialize(to\_serialize.sha256\_hash);

out.reserve(path\_part.size() + size\_part.size() + action\_part.size() +

last\_write\_part.size() + action\_part.size());

out.insert(out.end(), path\_part.begin(), path\_part.end());

out.insert(out.end(), size\_part.begin(), size\_part.end());

out.insert(out.end(), action\_part.begin(), action\_part.end());

out.insert(out.end(), last\_write\_part.begin(), last\_write\_part.end());

out.insert(out.end(), sha\_part.begin(), sha\_part.end());

return out;

}

models::file\_frame::FileHeader Deserialize(

SerializedContainer::const\_iterator& start\_position,

to::To<models::file\_frame::FileHeader>){

auto path = Deserialize(start\_position, to::To<std::string>());

auto size = Deserialize(start\_position, to::To<uintmax\_t>());

auto action\_str = Deserialize(start\_position, to::To<std::string>());

auto last\_write\_epoch = Deserialize(start\_position, to::To<int64\_t>());

auto sha256 = Deserialize(start\_position, to::To<std::string>());

std::chrono::time\_point<std::chrono::system\_clock, std::chrono::milliseconds>

last\_write\_tp{std::chrono::milliseconds{last\_write\_epoch}};

return models::file\_frame::FileHeader{

path,

size,

Convert(action\_str, helpers::to::To<models::file\_frame::Action>()),

std::chrono::file\_clock::from\_sys(last\_write\_tp),

sha256

};

}

SerializedContainer Serialize(

const models::file\_frame::FragmentHeader& to\_serialize) {

SerializedContainer out;

auto part = Serialize(to\_serialize.part);

auto parts = Serialize(to\_serialize.parts);

auto part\_sha256\_hash = Serialize(to\_serialize.part\_sha256\_hash);

auto write\_position = Serialize(to\_serialize.write\_position);

out.reserve(part.size() + parts.size() + part\_sha256\_hash.size() +

write\_position.size());

out.insert(out.end(), part.begin(), part.end());

out.insert(out.end(), parts.begin(), parts.end());

out.insert(out.end(), part\_sha256\_hash.begin(), part\_sha256\_hash.end());

out.insert(out.end(), write\_position.begin(), write\_position.end());

return out;

}

models::file\_frame::FragmentHeader Deserialize(

SerializedContainer::const\_iterator& start\_position,

to::To<models::file\_frame::FragmentHeader>){

auto part = Deserialize(start\_position, to::To<uint64\_t>());

auto parts = Deserialize(start\_position, to::To<uint64\_t>());

auto part\_sha256\_hash = Deserialize(start\_position, to::To<std::string>());

auto write\_position = Deserialize(start\_position, to::To<uint64\_t>());

return models::file\_frame::FragmentHeader{

part,

parts,

part\_sha256\_hash,

write\_position

};

}

SerializedContainer Serialize(

const models::file\_frame::FileFragment& to\_serialize) {

SerializedContainer out;

auto file\_header = Serialize(to\_serialize.file\_header);

auto fragment\_header = Serialize(to\_serialize.fragment\_header);

auto idempotency\_token = Serialize(to\_serialize.idempotency\_token);

auto payload = Serialize(to\_serialize.payload);

out.reserve(file\_header.size() + fragment\_header.size() + idempotency\_token.size() + payload.size());

out.insert(out.end(), file\_header.begin(), file\_header.end());

out.insert(out.end(), fragment\_header.begin(), fragment\_header.end());

out.insert(out.end(), idempotency\_token.begin(), idempotency\_token.end());

out.insert(out.end(), payload.begin(), payload.end());

return out;

}

models::file\_frame::FileFragment Deserialize(

SerializedContainer::const\_iterator& start\_position,

to::To<models::file\_frame::FileFragment>){

auto file\_header = Deserialize(start\_position, to::To<models::file\_frame::FileHeader>());

auto fragment\_header = Deserialize(start\_position, to::To<models::file\_frame::FragmentHeader>());

auto idempotency\_token = Deserialize(start\_position, to::To<std::string>());

auto payload = Deserialize(start\_position, to::To<models::file\_frame::Payload>());

return models::file\_frame::FileFragment{

file\_header,

fragment\_header,

idempotency\_token,

payload

};

}

} // namespace helpers::serialization

Б.12 Текст программы «src\models\idempotency\_token.hpp»

#pragma once

#include <string>

#include <models/file\_frame.hpp>

namespace models::idempotency\_token {

std::string GetToken(const file\_frame::FileHeader& file\_header,

const file\_frame::FragmentHeader& fragment\_header);

}

Б.13 Текст программы «src\models\idempotency\_token.cpp»

#include "idempotency\_token.hpp"

#include <fmt/core.h>

namespace models::idempotency\_token {

std::string GetToken(const file\_frame::FileHeader& file\_header,

const file\_frame::FragmentHeader& fragment\_header) {

return fmt::format("{}-{}-{}-{}-{}", file\_header.path,

file\_header.last\_write\_time.time\_since\_epoch().count(),

file\_header.size, fragment\_header.part,

fragment\_header.parts);

}

} // namespace models::idempotency\_token

Б.14 Текст программы «src\models\udp\_buffer.hpp»

#pragma once

#include <atomic>

#include <exception>

#include <memory>

#include <mutex>

#include <vector>

namespace models::udp\_buffer {

struct DataBuffer {

DataBuffer(int \_last\_datagram\_size, std::vector<char>&& \_buffer)

: last\_datagram\_size(\_last\_datagram\_size), buffer(\_buffer) {}

int last\_datagram\_size;

std::vector<char> buffer;

};

class BufferQueue {

public:

class AllBuffersLocked : std::runtime\_error {

using std::runtime\_error::runtime\_error;

};

BufferQueue(int queue\_size, int buffer\_size);

std::unique\_ptr<DataBuffer> AquireBuffer();

void ReleaseBuffer(std::unique\_ptr<DataBuffer> buffer\_to\_release);

private:

int buffer\_size\_;

std::vector<std::unique\_ptr<DataBuffer>> free\_buffers\_;

std::mutex mutex\_;

};

} // namespace models::udp\_buffer

Б.15 Текст программы «src\models\udp\_buffer.cpp»

#include "udp\_buffer.hpp"

#include <utils/logging.hpp>

namespace models::udp\_buffer {

BufferQueue::BufferQueue(int queue\_size, int buffer\_size) {

buffer\_size\_ = buffer\_size;

free\_buffers\_.reserve(queue\_size);

for (int i = 0; i < queue\_size; i++) {

free\_buffers\_.emplace\_back(

std::make\_unique<DataBuffer>(0, std::vector<char>(buffer\_size\_))

);

}

}

std::unique\_ptr<DataBuffer> BufferQueue::AquireBuffer() {

std::lock\_guard lock(mutex\_);

if(free\_buffers\_.size() == 0){

buffer\_size\_+=1;

LOG\_INFO() << "Added extra buffer, now total: " << buffer\_size\_ << '\n';

free\_buffers\_.emplace\_back(

std::make\_unique<DataBuffer>(0, std::vector<char>(buffer\_size\_))

);

}

auto buffer = std::move(free\_buffers\_.back());

free\_buffers\_.pop\_back();

return buffer;

}

void BufferQueue::ReleaseBuffer(std::unique\_ptr<DataBuffer> buffer\_to\_release){

std::lock\_guard lock(mutex\_);

free\_buffers\_.push\_back(std::move(buffer\_to\_release));

}

} // namespace models::udp\_buffer

Б.16 Текст программы «src\models\udp\_frame\_queue.hpp»

#pragma once

#include <boost/asio.hpp>

#include <models/udp\_buffer.hpp>

#include <modules/asio\_job\_queue.hpp>

#include <vector>

namespace networking::frame\_queue {

namespace ip = boost::asio::ip;

class FrameQueue {

public:

FrameQueue(int workers\_count) : strand\_(io\_service\_), pool\_(workers\_count) {}

void Add(models::udp\_buffer::DataBuffer& frame\_buffer);

private:

boost::asio::io\_service io\_service\_;

boost::asio::io\_service::strand strand\_;

modules::asio\_job\_queue::AsioJobQueue pool\_;

std::mutex mutex\_;

};

} // namespace networking::frame\_queue

Б.17 Текст программы «src\models\udp\_frame\_queue.cpp»

#include "udp\_frame\_queue.hpp"

#include <fmt/core.h>

namespace networking::frame\_queue {

void FrameQueue::Add(models::udp\_buffer::DataBuffer& frame\_buffer) {

// Perform pre\_process checks

// Structural\_binding

pool\_.Shedule([&frame\_buffer]() {

std::cout << "FrameQueue" << frame\_buffer.last\_datagram\_size;

std::cout << fmt::format("Processed Frame with size: {}\n",

frame\_buffer.last\_datagram\_size);

});

}

} // namespace networking::frame\_queue

Б.18 Текст программы «src\helpers\serialization.hpp»

#pragma once

#include <helpers/to.hpp>

#include <string>

#include <tuple>

#include <vector>

namespace helpers::serialization {

static const int kSizeBytes = 8;

using SerializedContainer = std::vector<char>;

SerializedContainer Serialize(const int64\_t& to\_serialize);

int64\_t Deserialize(SerializedContainer::const\_iterator& start\_position, to::To<int64\_t>);

SerializedContainer Serialize(const uint64\_t& to\_serialize);

uint64\_t Deserialize(SerializedContainer::const\_iterator& start\_position,

to::To<uint64\_t>);

SerializedContainer Serialize(const std::string& to\_serialize);

std::string Deserialize(SerializedContainer::const\_iterator& start\_position,

to::To<std::string>);

SerializedContainer Serialize(const std::vector<char>& to\_serialize);

std::vector<char> Deserialize(

SerializedContainer::const\_iterator& start\_position,

to::To<std::vector<char>>);

} // namespace helpers::serialization

Б.19 Текст программы «src\helpers\serialization.cpp»

#include "serialization.hpp"

namespace helpers::serialization {

namespace {

template<typename IntType>

SerializedContainer SerializeInt(const IntType& to\_serialize) {

SerializedContainer out;

out.reserve(sizeof(IntType));

for (int offset = sizeof(IntType) \* 8 - 8; offset >= 0; offset -= 8) {

out.push\_back(to\_serialize & (0xFFULL << (offset)));

}

return out;

}

template<typename IntType>

IntType DeserializeInt(SerializedContainer::const\_iterator& start\_position) {

IntType out = 0;

const auto end = start\_position + sizeof(IntType);

for (auto it = start\_position; it != end; it++) {

out = (out << 8) | \*it;

}

start\_position = end;

return out;

}

template<typename Container>

SerializedContainer SerializeContainer(const Container& to\_serialize) {

SerializedContainer container;

container.reserve(to\_serialize.size() + kSizeBytes);

auto size\_part = Serialize(to\_serialize.size());

container.insert(container.begin(), size\_part.begin(), size\_part.end());

container.insert(container.end(), to\_serialize.begin(), to\_serialize.end());

return container;

}

template<typename Container>

Container DeserializeContainer(SerializedContainer::const\_iterator& start\_position){

const auto size = Deserialize(start\_position, to::To<std::uint64\_t>());

const auto string\_start = start\_position + kSizeBytes;

start\_position = string\_start + size;

Container out(string\_start, string\_start + size);

return out;

}

}

SerializedContainer Serialize(const int64\_t& to\_serialize) {

return SerializeInt<int64\_t>(to\_serialize);

}

int64\_t Deserialize(SerializedContainer::const\_iterator& start\_position,

to::To<int64\_t>) {

return DeserializeInt<int64\_t>(start\_position);

}

SerializedContainer Serialize(const uint64\_t& to\_serialize) {

return SerializeInt<uint64\_t>(to\_serialize);

}

uint64\_t Deserialize(SerializedContainer::const\_iterator& start\_position,

to::To<uint64\_t>) {

return DeserializeInt<uint64\_t>(start\_position);

}

SerializedContainer Serialize(const std::string& to\_serialize) {

return SerializeContainer<std::string>(to\_serialize);

}

std::string Deserialize(SerializedContainer::const\_iterator& start\_position,

to::To<std::string>){

return DeserializeContainer<std::string>(start\_position);

}

SerializedContainer Serialize(const std::vector<char>& to\_serialize) {

return SerializeContainer<std::vector<char>>(to\_serialize);

}

std::vector<char> Deserialize(SerializedContainer::const\_iterator& start\_position,

to::To<std::vector<char>>){

return DeserializeContainer<std::vector<char>>(start\_position);

}

}

Б.20 Текст программы «src\helpers\to.hpp»

#pragma once

#include <stdexcept>

namespace helpers::to {

template <typename T>

struct To {};

class ConvertError : std::runtime\_error {

using std::runtime\_error::runtime\_error;

};

class SerializationError : std::runtime\_error {

using std::runtime\_error::runtime\_error;

};

} // namespace helpers::to

Б.21 Текст программы «src\utils\calculate\_hash.hpp»

#pragma once

#include <string>

#include <vector>

#include <fstream>

namespace utils::calculate\_hash{

std::string Calculate(std::ifstream& file);

std::string Calculate(const std::vector<char>& data);

} // namespace utils::calculate\_hash

Б.22 Текст программы «src\utils\calculate\_hash.cpp»

#include "calculate\_hash.hpp"

#include <utils/sha256.h>

namespace utils::calculate\_hash {

std::string Calculate(std::ifstream& file) {

return sha256(file);

}

std::string Calculate(const std::vector<char>& data) {

return sha256(data);

}

} // namespace utils::calculate\_hash

Б.23 Текст программы «src\utils\call\_each.hpp»

#pragma once

#include <boost/asio.hpp>

#include <functional>

namespace utils::call\_each {

class CallEach {

public:

using Handler = std::function<void()>;

CallEach(Handler handler, std::chrono::seconds delay) : handler\_(handler),

check\_each\_(delay), thread\_([this] {

Update();

io\_service\_.run();

}) {}

~CallEach() {

io\_service\_.stop();

try {

thread\_.join();

} catch (const std::exception&) {

}

}

private:

void Update() {

timer\_.expires\_from\_now(check\_each\_);

timer\_.async\_wait([this](const boost::system::error\_code& e) {

if(!e || e.value() != boost::asio::error::operation\_aborted){

handler\_();

}

Update();

});

}

Handler handler\_;

boost::asio::io\_service io\_service\_;

boost::asio::io\_service::work work\_{io\_service\_};

boost::asio::steady\_timer timer\_{io\_service\_};

std::chrono::seconds check\_each\_;

std::thread thread\_;

};

} // namespace utils::call\_each

Б.24 Текст программы «src\utils\logging.hpp»

#pragma once

#include <iostream>

#define LOG\_TARGET std::cout

#define \_LOG\_DEBUG true

#define LOG\_DEBUG() if(\_LOG\_DEBUG) LOG\_TARGET

#define \_LOG\_INFO true

#define LOG\_INFO() if(\_LOG\_INFO) LOG\_TARGET

#define \_LOG\_WARNING true

#define LOG\_WARNING() if(\_LOG\_WARNING) LOG\_TARGET

Б.25 Текст программы «src\utils\sha256.h»

#pragma once

#include <string>

#include <stdint.h>

namespace utils::sha256 {

class SHA256

{

public:

enum { BlockSize = 512 / 8, HashBytes = 32 };

SHA256();

std::string operator()(const void\* data, size\_t numBytes);

std::string operator()(const std::string& text);

void add(const void\* data, size\_t numBytes);

std::string getHash();

void getHash(unsigned char buffer[HashBytes]);

void reset();

private:

void processBlock(const void\* data);

void processBuffer();

uint64\_t m\_numBytes;

size\_t m\_bufferSize;

uint8\_t m\_buffer[BlockSize];

enum { HashValues = HashBytes / 4 };

uint32\_t m\_hash[HashValues];

};

}

Б.26 Текст программы «src\utils\sha256.cpp»

#include "sha256.h"

namespace utils::sha256 {

/// same as reset()

SHA256::SHA256()

{

reset();

}

/// restart

void SHA256::reset()

{

m\_numBytes = 0;

m\_bufferSize = 0;

// according to RFC 1321

m\_hash[0] = 0x6a09e667;

m\_hash[1] = 0xbb67ae85;

m\_hash[2] = 0x3c6ef372;

m\_hash[3] = 0xa54ff53a;

m\_hash[4] = 0x510e527f;

m\_hash[5] = 0x9b05688c;

m\_hash[6] = 0x1f83d9ab;

m\_hash[7] = 0x5be0cd19;

}

namespace

{

inline uint32\_t rotate(uint32\_t a, uint32\_t c)

{

return (a >> c) | (a << (32 - c));

}

inline uint32\_t swap(uint32\_t x)

{

#if defined(\_\_GNUC\_\_) || defined(\_\_clang\_\_)

return \_\_builtin\_bswap32(x);

#endif

#ifdef MSC\_VER

return \_byteswap\_ulong(x);

#endif

return (x >> 24) |

((x >> 8) & 0x0000FF00) |

((x << 8) & 0x00FF0000) |

(x << 24);

}

// mix functions for processBlock()

inline uint32\_t f1(uint32\_t e, uint32\_t f, uint32\_t g)

{

uint32\_t term1 = rotate(e, 6) ^ rotate(e, 11) ^ rotate(e, 25);

uint32\_t term2 = (e & f) ^ (~e & g); //(g ^ (e & (f ^ g)))

return term1 + term2;

}

inline uint32\_t f2(uint32\_t a, uint32\_t b, uint32\_t c)

{

uint32\_t term1 = rotate(a, 2) ^ rotate(a, 13) ^ rotate(a, 22);

uint32\_t term2 = ((a | b) & c) | (a & b); //(a & (b ^ c)) ^ (b & c);

return term1 + term2;

}

}

/// process 64 bytes

void SHA256::processBlock(const void\* data)

{

// get last hash

uint32\_t a = m\_hash[0];

uint32\_t b = m\_hash[1];

uint32\_t c = m\_hash[2];

uint32\_t d = m\_hash[3];

uint32\_t e = m\_hash[4];

uint32\_t f = m\_hash[5];

uint32\_t g = m\_hash[6];

uint32\_t h = m\_hash[7];

// data represented as 16x 32-bit words

const uint32\_t\* input = (uint32\_t\*) data;

// convert to big endian

uint32\_t words[64];

int i;

for (i = 0; i < 16; i++)

#if defined(\_\_BYTE\_ORDER) && (\_\_BYTE\_ORDER != 0) && (\_\_BYTE\_ORDER == \_\_BIG\_ENDIAN)

words[i] = input[i];

#else

words[i] = swap(input[i]);

#endif

uint32\_t x,y; // temporaries

// first round

x = h + f1(e,f,g) + 0x428a2f98 + words[ 0]; y = f2(a,b,c); d += x; h = x + y;

x = g + f1(d,e,f) + 0x71374491 + words[ 1]; y = f2(h,a,b); c += x; g = x + y;

x = f + f1(c,d,e) + 0xb5c0fbcf + words[ 2]; y = f2(g,h,a); b += x; f = x + y;

x = e + f1(b,c,d) + 0xe9b5dba5 + words[ 3]; y = f2(f,g,h); a += x; e = x + y;

x = d + f1(a,b,c) + 0x3956c25b + words[ 4]; y = f2(e,f,g); h += x; d = x + y;

x = c + f1(h,a,b) + 0x59f111f1 + words[ 5]; y = f2(d,e,f); g += x; c = x + y;

x = b + f1(g,h,a) + 0x923f82a4 + words[ 6]; y = f2(c,d,e); f += x; b = x + y;

x = a + f1(f,g,h) + 0xab1c5ed5 + words[ 7]; y = f2(b,c,d); e += x; a = x + y;

// secound round

x = h + f1(e,f,g) + 0xd807aa98 + words[ 8]; y = f2(a,b,c); d += x; h = x + y;

x = g + f1(d,e,f) + 0x12835b01 + words[ 9]; y = f2(h,a,b); c += x; g = x + y;

x = f + f1(c,d,e) + 0x243185be + words[10]; y = f2(g,h,a); b += x; f = x + y;

x = e + f1(b,c,d) + 0x550c7dc3 + words[11]; y = f2(f,g,h); a += x; e = x + y;

x = d + f1(a,b,c) + 0x72be5d74 + words[12]; y = f2(e,f,g); h += x; d = x + y;

x = c + f1(h,a,b) + 0x80deb1fe + words[13]; y = f2(d,e,f); g += x; c = x + y;

x = b + f1(g,h,a) + 0x9bdc06a7 + words[14]; y = f2(c,d,e); f += x; b = x + y;

x = a + f1(f,g,h) + 0xc19bf174 + words[15]; y = f2(b,c,d); e += x; a = x + y;

// extend to 24 words

for (; i < 24; i++)

words[i] = words[i-16] +

(rotate(words[i-15], 7) ^ rotate(words[i-15], 18) ^ (words[i-15] >> 3)) +

words[i-7] +

(rotate(words[i- 2], 17) ^ rotate(words[i- 2], 19) ^ (words[i- 2] >> 10));

// third round

x = h + f1(e,f,g) + 0xe49b69c1 + words[16]; y = f2(a,b,c); d += x; h = x + y;

x = g + f1(d,e,f) + 0xefbe4786 + words[17]; y = f2(h,a,b); c += x; g = x + y;

x = f + f1(c,d,e) + 0x0fc19dc6 + words[18]; y = f2(g,h,a); b += x; f = x + y;

x = e + f1(b,c,d) + 0x240ca1cc + words[19]; y = f2(f,g,h); a += x; e = x + y;

x = d + f1(a,b,c) + 0x2de92c6f + words[20]; y = f2(e,f,g); h += x; d = x + y;

x = c + f1(h,a,b) + 0x4a7484aa + words[21]; y = f2(d,e,f); g += x; c = x + y;

x = b + f1(g,h,a) + 0x5cb0a9dc + words[22]; y = f2(c,d,e); f += x; b = x + y;

x = a + f1(f,g,h) + 0x76f988da + words[23]; y = f2(b,c,d); e += x; a = x + y;

// extend to 32 words

for (; i < 32; i++)

words[i] = words[i-16] +

(rotate(words[i-15], 7) ^ rotate(words[i-15], 18) ^ (words[i-15] >> 3)) +

words[i-7] +

(rotate(words[i- 2], 17) ^ rotate(words[i- 2], 19) ^ (words[i- 2] >> 10));

// fourth round

x = h + f1(e,f,g) + 0x983e5152 + words[24]; y = f2(a,b,c); d += x; h = x + y;

x = g + f1(d,e,f) + 0xa831c66d + words[25]; y = f2(h,a,b); c += x; g = x + y;

x = f + f1(c,d,e) + 0xb00327c8 + words[26]; y = f2(g,h,a); b += x; f = x + y;

x = e + f1(b,c,d) + 0xbf597fc7 + words[27]; y = f2(f,g,h); a += x; e = x + y;

x = d + f1(a,b,c) + 0xc6e00bf3 + words[28]; y = f2(e,f,g); h += x; d = x + y;

x = c + f1(h,a,b) + 0xd5a79147 + words[29]; y = f2(d,e,f); g += x; c = x + y;

x = b + f1(g,h,a) + 0x06ca6351 + words[30]; y = f2(c,d,e); f += x; b = x + y;

x = a + f1(f,g,h) + 0x14292967 + words[31]; y = f2(b,c,d); e += x; a = x + y;

// extend to 40 words

for (; i < 40; i++)

words[i] = words[i-16] +

(rotate(words[i-15], 7) ^ rotate(words[i-15], 18) ^ (words[i-15] >> 3)) +

words[i-7] +

(rotate(words[i- 2], 17) ^ rotate(words[i- 2], 19) ^ (words[i- 2] >> 10));

// fifth round

x = h + f1(e,f,g) + 0x27b70a85 + words[32]; y = f2(a,b,c); d += x; h = x + y;

x = g + f1(d,e,f) + 0x2e1b2138 + words[33]; y = f2(h,a,b); c += x; g = x + y;

x = f + f1(c,d,e) + 0x4d2c6dfc + words[34]; y = f2(g,h,a); b += x; f = x + y;

x = e + f1(b,c,d) + 0x53380d13 + words[35]; y = f2(f,g,h); a += x; e = x + y;

x = d + f1(a,b,c) + 0x650a7354 + words[36]; y = f2(e,f,g); h += x; d = x + y;

x = c + f1(h,a,b) + 0x766a0abb + words[37]; y = f2(d,e,f); g += x; c = x + y;

x = b + f1(g,h,a) + 0x81c2c92e + words[38]; y = f2(c,d,e); f += x; b = x + y;

x = a + f1(f,g,h) + 0x92722c85 + words[39]; y = f2(b,c,d); e += x; a = x + y;

// extend to 48 words

for (; i < 48; i++)

words[i] = words[i-16] +

(rotate(words[i-15], 7) ^ rotate(words[i-15], 18) ^ (words[i-15] >> 3)) +

words[i-7] +

(rotate(words[i- 2], 17) ^ rotate(words[i- 2], 19) ^ (words[i- 2] >> 10));

// sixth round

x = h + f1(e,f,g) + 0xa2bfe8a1 + words[40]; y = f2(a,b,c); d += x; h = x + y;

x = g + f1(d,e,f) + 0xa81a664b + words[41]; y = f2(h,a,b); c += x; g = x + y;

x = f + f1(c,d,e) + 0xc24b8b70 + words[42]; y = f2(g,h,a); b += x; f = x + y;

x = e + f1(b,c,d) + 0xc76c51a3 + words[43]; y = f2(f,g,h); a += x; e = x + y;

x = d + f1(a,b,c) + 0xd192e819 + words[44]; y = f2(e,f,g); h += x; d = x + y;

x = c + f1(h,a,b) + 0xd6990624 + words[45]; y = f2(d,e,f); g += x; c = x + y;

x = b + f1(g,h,a) + 0xf40e3585 + words[46]; y = f2(c,d,e); f += x; b = x + y;

x = a + f1(f,g,h) + 0x106aa070 + words[47]; y = f2(b,c,d); e += x; a = x + y;

// extend to 56 words

for (; i < 56; i++)

words[i] = words[i-16] +

(rotate(words[i-15], 7) ^ rotate(words[i-15], 18) ^ (words[i-15] >> 3)) +

words[i-7] +

(rotate(words[i- 2], 17) ^ rotate(words[i- 2], 19) ^ (words[i- 2] >> 10));

// seventh round

x = h + f1(e,f,g) + 0x19a4c116 + words[48]; y = f2(a,b,c); d += x; h = x + y;

x = g + f1(d,e,f) + 0x1e376c08 + words[49]; y = f2(h,a,b); c += x; g = x + y;

x = f + f1(c,d,e) + 0x2748774c + words[50]; y = f2(g,h,a); b += x; f = x + y;

x = e + f1(b,c,d) + 0x34b0bcb5 + words[51]; y = f2(f,g,h); a += x; e = x + y;

x = d + f1(a,b,c) + 0x391c0cb3 + words[52]; y = f2(e,f,g); h += x; d = x + y;

x = c + f1(h,a,b) + 0x4ed8aa4a + words[53]; y = f2(d,e,f); g += x; c = x + y;

x = b + f1(g,h,a) + 0x5b9cca4f + words[54]; y = f2(c,d,e); f += x; b = x + y;

x = a + f1(f,g,h) + 0x682e6ff3 + words[55]; y = f2(b,c,d); e += x; a = x + y;

// extend to 64 words

for (; i < 64; i++)

words[i] = words[i-16] +

(rotate(words[i-15], 7) ^ rotate(words[i-15], 18) ^ (words[i-15] >> 3)) +

words[i-7] +

(rotate(words[i- 2], 17) ^ rotate(words[i- 2], 19) ^ (words[i- 2] >> 10));

// eigth round

x = h + f1(e,f,g) + 0x748f82ee + words[56]; y = f2(a,b,c); d += x; h = x + y;

x = g + f1(d,e,f) + 0x78a5636f + words[57]; y = f2(h,a,b); c += x; g = x + y;

x = f + f1(c,d,e) + 0x84c87814 + words[58]; y = f2(g,h,a); b += x; f = x + y;

x = e + f1(b,c,d) + 0x8cc70208 + words[59]; y = f2(f,g,h); a += x; e = x + y;

x = d + f1(a,b,c) + 0x90befffa + words[60]; y = f2(e,f,g); h += x; d = x + y;

x = c + f1(h,a,b) + 0xa4506ceb + words[61]; y = f2(d,e,f); g += x; c = x + y;

x = b + f1(g,h,a) + 0xbef9a3f7 + words[62]; y = f2(c,d,e); f += x; b = x + y;

x = a + f1(f,g,h) + 0xc67178f2 + words[63]; y = f2(b,c,d); e += x; a = x + y;

// update hash

m\_hash[0] += a;

m\_hash[1] += b;

m\_hash[2] += c;

m\_hash[3] += d;

m\_hash[4] += e;

m\_hash[5] += f;

m\_hash[6] += g;

m\_hash[7] += h;

}

/// add arbitrary number of bytes

void SHA256::add(const void\* data, size\_t numBytes)

{

const uint8\_t\* current = (const uint8\_t\*) data;

if (m\_bufferSize > 0)

{

while (numBytes > 0 && m\_bufferSize < BlockSize)

{

m\_buffer[m\_bufferSize++] = \*current++;

numBytes--;

}

}

// full buffer

if (m\_bufferSize == BlockSize)

{

processBlock(m\_buffer);

m\_numBytes += BlockSize;

m\_bufferSize = 0;

}

// no more data ?

if (numBytes == 0)

return;

// process full blocks

while (numBytes >= BlockSize)

{

processBlock(current);

current += BlockSize;

m\_numBytes += BlockSize;

numBytes -= BlockSize;

}

// keep remaining bytes in buffer

while (numBytes > 0)

{

m\_buffer[m\_bufferSize++] = \*current++;

numBytes--;

}

}

/// process final block, less than 64 bytes

void SHA256::processBuffer()

{

// the input bytes are considered as bits strings, where the first bit is the most significant bit of the byte

// - append "1" bit to message

// - append "0" bits until message length in bit mod 512 is 448

// - append length as 64 bit integer

// number of bits

size\_t paddedLength = m\_bufferSize \* 8;

// plus one bit set to 1 (always appended)

paddedLength++;

// number of bits must be (numBits % 512) = 448

size\_t lower11Bits = paddedLength & 511;

if (lower11Bits <= 448)

paddedLength += 448 - lower11Bits;

else

paddedLength += 512 + 448 - lower11Bits;

// convert from bits to bytes

paddedLength /= 8;

// only needed if additional data flows over into a second block

unsigned char extra[BlockSize];

// append a "1" bit, 128 => binary 10000000

if (m\_bufferSize < BlockSize)

m\_buffer[m\_bufferSize] = 128;

else

extra[0] = 128;

size\_t i;

for (i = m\_bufferSize + 1; i < BlockSize; i++)

m\_buffer[i] = 0;

for (; i < paddedLength; i++)

extra[i - BlockSize] = 0;

// add message length in bits as 64 bit number

uint64\_t msgBits = 8 \* (m\_numBytes + m\_bufferSize);

// find right position

unsigned char\* addLength;

if (paddedLength < BlockSize)

addLength = m\_buffer + paddedLength;

else

addLength = extra + paddedLength - BlockSize;

// must be big endian

\*addLength++ = (unsigned char)((msgBits >> 56) & 0xFF);

\*addLength++ = (unsigned char)((msgBits >> 48) & 0xFF);

\*addLength++ = (unsigned char)((msgBits >> 40) & 0xFF);

\*addLength++ = (unsigned char)((msgBits >> 32) & 0xFF);

\*addLength++ = (unsigned char)((msgBits >> 24) & 0xFF);

\*addLength++ = (unsigned char)((msgBits >> 16) & 0xFF);

\*addLength++ = (unsigned char)((msgBits >> 8) & 0xFF);

\*addLength = (unsigned char)( msgBits & 0xFF);

// process blocks

processBlock(m\_buffer);

// flowed over into a second block ?

if (paddedLength > BlockSize)

processBlock(extra);

}

/// return latest hash as 64 hex characters

std::string SHA256::getHash()

{

// compute hash (as raw bytes)

unsigned char rawHash[HashBytes];

getHash(rawHash);

// convert to hex string

std::string result;

result.reserve(2 \* HashBytes);

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

{

static const char dec2hex[16+1] = "0123456789abcdef";

result += dec2hex[(rawHash[i] >> 4) & 15];

result += dec2hex[ rawHash[i] & 15];

}

return result;

}

/// return latest hash as bytes

void SHA256::getHash(unsigned char buffer[SHA256::HashBytes])

{

// save old hash if buffer is partially filled

uint32\_t oldHash[HashValues];

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

oldHash[i] = m\_hash[i];

// process remaining bytes

processBuffer();

unsigned char\* current = buffer;

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

{

\*current++ = (m\_hash[i] >> 24) & 0xFF;

\*current++ = (m\_hash[i] >> 16) & 0xFF;

\*current++ = (m\_hash[i] >> 8) & 0xFF;

\*current++ = m\_hash[i] & 0xFF;

// restore old hash

m\_hash[i] = oldHash[i];

}

}

/// compute SHA256 of a memory block

std::string SHA256::operator()(const void\* data, size\_t numBytes)

{

reset();

add(data, numBytes);

return getHash();

}

/// compute SHA256 of a string, excluding final zero

std::string SHA256::operator()(const std::string& text)

{

reset();

add(text.c\_str(), text.size());

return getHash();

}

}

Б.27 Текст программы «test\_snippets\pow\_file\_watcher.cpp»

#include <chrono>

#include <string>

#include <thread>

#include <config/type/fs\_watcher.hpp>

#include <modules/fs\_watcher.hpp>

#include <utils/logging.hpp>

#include <fmt/core.h>

int main() {

auto watcher\_cfg = config::filesystem\_watcher::Default();

auto watcher = modules::fs\_watcher::FileWatcher(watcher\_cfg, [](const modules::fs\_watcher::FileDiscription& discr, modules::fs\_watcher::FileAction act){

const static auto act\_to\_string = [](modules::fs\_watcher::FileAction& act) {

switch(act){

case modules::fs\_watcher::FileAction::kCreated:

return "Created";

case modules::fs\_watcher::FileAction::kModified:

return "Modified";

case modules::fs\_watcher::FileAction::kDeleted:

return "Deleted";

}

};

LOG\_INFO() << fmt::format("Update> Path: {}, Size: {}, Action: {}\n", discr.path.string(), discr.size, act\_to\_string(act));

});

watcher.List([](const modules::fs\_watcher::FileDiscription& discr){

LOG\_INFO() << fmt::format("INIT> Path: {}, Size: {}\n", discr.path.string(), discr.size);

});

std::this\_thread::sleep\_for(std::chrono::minutes(5));

return 0;

}

Б.28 Текст программы «test\_snippets\pow\_send\_receive.cpp»

#include <chrono>

#include <string>

#include <thread>

#include <config/type/udp\_receiver.hpp>

#include <config/type/udp\_sender.hpp>

#include <networking/udp\_receiver.hpp>

#include <networking/udp\_sender.hpp>

#include <utils/logging.hpp>

int main() {

auto receiver\_cfg = config::udp\_receiver::Default();

auto sender\_cfg = config::udp\_sender::Default();

auto receiver = networking::udp\_receiver::UdpReceiver(

receiver\_cfg, [](models::udp\_buffer::DataBuffer& data) {

LOG\_INFO() << "RECEIVER\_CALLBACK: "<< std::string(

std::begin(data.buffer),

std::begin(data.buffer) + data.last\_datagram\_size)

<< "\n";

});

auto sender = networking::udp\_sender::UdpSender(sender\_cfg);

receiver.Start();

sender.Start();

auto payload = std::string("Some data");

auto data = std::vector<char>(std::begin(payload), std::end(payload));

sender.Send(std::make\_unique<networking::udp\_sender::UdpSender::Package>(

networking::udp\_sender::UdpSender::Package{std::move(data), 0}));

using namespace std::chrono\_literals;

std::this\_thread::sleep\_for(2000ms);

return 0;

}

Б.29 Текст программы «test\_snippets\pow\_serialization.cpp»

#include <chrono>

#include <string>

#include <helpers/serialization.hpp>

#include <utils/logging.hpp>

int main() {

const auto serialized = helpers::serialization::Serialize("SErialize Test");

const auto deserialized = helpers::serialization::Deserialize(serialized.begin(), helpers::serialization::To<std::string>());

LOG\_INFO() << deserialized;

}

Б.30 Текст программы «src\config\type\udp\_sender.hpp»

#pragma once

#include <boost/asio/ip/address.hpp>

#include <boost/asio/ip/udp.hpp>

#include <config/storage.hpp>

namespace config::udp\_sender {

namespace ip = boost::asio::ip;

struct Configuration {

ip::udp ip\_version;

ip::address target\_ip;

ip::port\_type target\_port;

int retry\_count;

};

Configuration Get(const ::config::storage::Storage& storage);

Configuration Default();

}

Б.31 Текст программы «src\config\type\udp\_receiver.hpp»

#pragma once

#include <optional>

#include <boost/asio/ip/address.hpp>

#include <boost/asio/ip/udp.hpp>

#include <config/storage.hpp>

namespace config::udp\_receiver {

namespace ip = boost::asio::ip;

struct Configuration {

ip::udp ip\_version; // udp::v4()

std::optional<ip::address> target\_ip; //ip::address\_v4::any()

ip::port\_type target\_port;

int datagram\_max\_size;

int buffer\_count;

int receive\_threads;

};

Configuration Get(const ::config::storage::Storage& storage);

Configuration Default();

}

Б.32 Текст программы «src\config\type\fs\_watcher.hpp»

#pragma once

#include <chrono>

#include <string>

#include <config/storage.hpp>

namespace config::filesystem\_watcher {

struct Configuration {

std::chrono::seconds check\_folder\_each;

std::string folder\_path;

uint64\_t max\_fragment\_size;

};

Configuration Get(const ::config::storage::Storage& storage);

Configuration Default();

}

Б.33 Текст файла конфигурации «CMakeLists.txt»

cmake\_minimum\_required(VERSION 3.14)

project(one-way-sync VERSION 0.1.0)

set(CMAKE\_CXX\_STANDARD 20)

include(FetchContent)

#Boost

set(BOOST\_ROOT "C:/CLI\_STUFF/mingw64/boost\_1\_78\_0/")

set(CMAKE\_INCLUDE\_PATH ${CMAKE\_INCLUDE\_PATH} "C:/CLI\_STUFF/mingw64/boost\_1\_78\_0/")

set(CMAKE\_LIBRARY\_PATH ${CMAKE\_LIBRARY\_PATH} "C:/CLI\_STUFF/mingw64/boost\_1\_78\_0/lib/")

set(Boost\_USE\_STATIC\_LIBS ON)

find\_package(Boost COMPONENTS system)

include\_directories(${Boost\_INCLUDE\_DIR})

#Threads

find\_package(Threads REQUIRED)

#FMT

FetchContent\_Declare(fmt

GIT\_REPOSITORY https://github.com/fmtlib/fmt.git

GIT\_TAG master

)

FetchContent\_MakeAvailable(fmt)

#JSON-parsing

FetchContent\_Declare(nlohmann\_json

GIT\_REPOSITORY https://github.com/nlohmann/json

GIT\_TAG master

)

FetchContent\_MakeAvailable(nlohmann\_json)

set (source\_dir "${PROJECT\_SOURCE\_DIR}/src/")

file (GLOB source\_files

"${source\_dir}/\*.cpp"

"${source\_dir}/config/type/\*.cpp"

"${source\_dir}/config/\*.cpp"

"${source\_dir}/helpers/\*.cpp"

"${source\_dir}/models/\*.cpp"

"${source\_dir}/modules/\*.cpp"

"${source\_dir}/networking/\*.cpp"

"${source\_dir}/utils/\*.cpp"

)

include\_directories(

"${source\_dir}/"

)

add\_compile\_options(-Wall -Wextra -pedantic -Werror -pthread)

add\_executable(one-way-sync ${source\_files})

target\_link\_libraries(one-way-sync ws2\_32)

target\_link\_libraries(one-way-sync Threads::Threads)

target\_link\_libraries(one-way-sync fmt::fmt-header-only)

target\_link\_libraries(one-way-sync nlohmann\_json::nlohmann\_json)