ОТЧЕТ

по курсовой работе

“IT-Школа Протей 2023-осень. Проектное задание C++”

Выполнил:
студент 3-го курса СПбГЭТУ “ЛЭТИ”
направления “Информационные системы и технологии”
Гаврилов Матвей

Санкт-Петербург
2023

Содержание

1. Постановка задачи

Написать простенький ЦОВ, который на вход будет получать HTTP запрос с указанием номера A и пытаться обработать такой “входящий вызов”.

Для каждого запроса приложение формирует свой уникальный идентификатор (CallID), который должен использоваться при записи в журналы и CDR.
Все запросы должны попадать в очередь длины N (задается в конфигурации), после чего должен отправляться HTTP ответ (http-response), ответ должен содержать CallID.

Приложение должно эмулировать ответ оператора или отбой через рандомное время в диапазоне от Rmin до Rmax (конфигурация). То есть, запросы из очереди постоянно выбираются и приложение пытается либо “распределить” вызов на свободного оператора, либо если все операторы заняты удерживать заявку в очереди в указанном диапазоне времени Rmin-Rmax.

Эмуляция распределения заключается в том, что оператор на случайное время (длительность такого занятия также нужно настраивать в конфигурации) переходит в состояние “Занят”. После завершения “общения”, оператор снова становится свободным и может быть использован для обработки новой заявки из очереди.
Если заявка на вызов так и не была обслужена за интервал Rmin-Rmax, вызов завершается отбоем с указанием причины (timeout).

Если на сервис приходит повторная заявка от номера, который уже стоит в очереди, то может быть 2 варианта поведения: ответ по http с сообщением об ошибке (already in queue) или завершение существующей заявки с указанием причины (call duplication) и постановкой новой заявки в конец очереди: оба варианта будут приняты, если можно будет настроить в конфигурации - будет только плюсом.

Количество операторов также должно настраиваться.

Если в момент поступления запроса очередь имеет максимально допустимый размер, то сервис должен отвечать ошибкой (http), CDR при этом всеравно формируется и в нем фиксируется особый статус “перегрузка” (overload).

По результатам работы должен формироваться CDR с указанием следующих параметров:

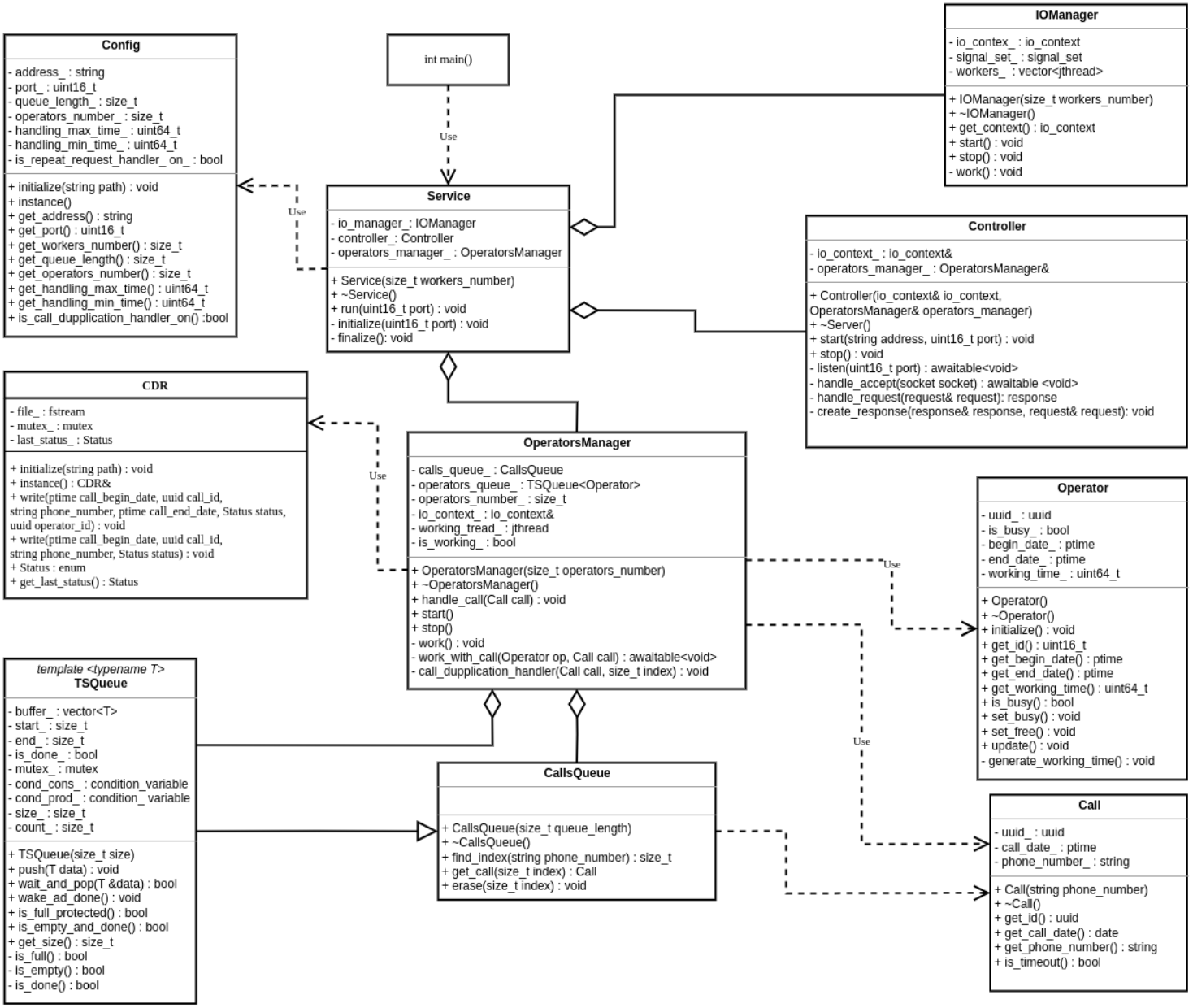
1. DT поступления вызова.
2. Идентификатор входящего вызова (Call ID)
3. Номер абонента A
4. DT завершения вызова
5. Статус вызова (OK или причина ошибки, например timeout)
6. DT ответа оператора (если был или пустое значение)
7. Идентификатор оператора (пустое значение если соединение не состоялось)
8. Длительность разговора (пустое значение если соединение не состоялось)

2. Архитектурное обоснование

Используемые технологии

1. Conan 1.61.0 - пакетный менеджер
2. CMake - система сборки
3. Boost - библиотека, используемая для работы с корутинами, протоколом HTTP, uuid-генераторами и posix_time инструментами.
4. spdlog - логгер
5. nlohman_json - json-парсер
6. GTest - библиотека, используемая для работы с unit-тестами

Диаграмма классов



Описание

1) Config

Является Singletone-классом, реализующим чтение и парсинг конфигурационного .json файла.

Обеспечивает гарантию корректной инициализации и доступа к параметрам конфигурации. В случае ошибки при открытии файла или некорректной конфигурации бросает исключение.

2) CDR

Является Sindletone-классом, реализующим журнал.

В случае ошибки при открытии или создании файла программу не завершает, но при каждой попытке записи делает предупреждение.

3) TSQueue

Шаблон класса потокобезопасной кольцевой очереди, являющейся реализацией паттерна producer-consumer.

Используется для хранения очереди свободных операторов.

4) CallsQueue

Является расширением класса TSQueue, добавляющим методы, необходимые для обработки повторных вызовов с одного номера телефона.

Используется для хранения очереди входящих звонков.

5) Call

Класс, используемый для хранения номера телефона, даты вызова, а так же генерации uuid. Помимо этого позволяет определить, вышло ли время на обработку вызова.

Инициализируется при создании объекта.

6) Operator

Класс, используемый для хранения даты ответа, генерации случайного времени обработки, генерации uuid.

Инициализируется вызовом специального метода.

7) IOManager

Реализует концепцию корутин. Создает io_context с указанным в конфигурации количеством worker-ов, после ожидает сигнала о завершении программы.

8) Controller

Используя io_context, созданный IOManager-ом, работает с протоколом HTTP. Принимает запросы, обрабатывает их, отправляет response. Создает объект класса Call и передает OperatorsManager-у для дальнейшей обработки.

9) OperatorsManager

Взаимодействует с очередями звонков и операторов. При связывании оператора и звонка использует io_context, созданный IOManager-ом, чтобы дождаться окончания обработки в асинхронном режиме.

10) Service

Запускает IOManager, Controller и OperatorsManager, ждет сигнала о завершении от IOManager-а, передает его Controller-у и OperatorsManager-у.

В случае некорректного запуска, работы или завершения ловит исключение и делает запись об ошибке в лог.

11) int main()

Инициализирует Config и CDR, запускает Service.

3. Документация

class CDR

Реализует журнал. Определяет возможные статусы вызовов в enum Status. Возможные статусы: OK, ALREADY_IN_QUEUE, CALL_DUPLICATION, TIMEOUT, OVERLOAD

Метод	**Описание**
void initialize(std::string path);	Открывает единственный файл журнала.
static CDR& instance();	Обеспечивает доступ к единственному экземпляру класса.
void write(const posix_time::ptime call_begin_date, const uuids::uuid call_id, const std::string phone_number, const posix_time::ptime call_end_date, const Status status, const posix_time::ptime answer_date, const uuids::uuid operator_id);	Делает запись в журнал в случае успешной обработки вызова. Потокобезопасен.
void write(const posix_time::ptime call_begin_date, const uuids::uuid call_id, const std::string phone_number, const Status status);	Делает запись в журнал в случае безуспешной обработки вызова. Потокобезопасен.
Status get_last_status() const;	Возвращает статус последнего обработанного вызова.

class Config

Парсит файл конфигурации. Дает доступ к параметрам конфигурации через геттеры.

Метод	**Описание**
void initialize(std::string path);	Открывает файл конфигурации и парсит его.

static Config& instance();	Обеспечивает доступ к единственному экземпляру класса.
std::string get_address() const	Возвращает IP-адрес.
std::uint16_t get_port() const	Возвращает порт.
size_t get_workers_number() const	Возвращает количество worker-ов для IOManager-a.
size_t get_queue_length() const;	Возвращает длину очереди вызовов.
size_t get_operators_number() const;	Возвращает количество работающих операторов.
uint64_t get_handling_min_time() const;	Возвращает минимальное время обработки звонка (в секундах).
uint64_t get_handling_max_time() const;	Возвращает максимальное время обработки звонка (в секундах)
bool is_call_dupplicaton_handler_on() const;	Показывает, включен ли обработчик повторных звонков с одного номера.

template <typename T> class TSQueue

Потокобезопасная кольцевая очередь.

Метод	**Описание**
TSQueue(size_t size)	Конструктор. Создает очередь размера size.
void push(T data)	Кладет данные в очередь. В случае если очередь полная или используется другим потоком, спит.
bool wait_and_pop(T &data)	Вытаскивает данные из очереди. В случае если очередь пустая, используется другим потоком или еще не закончила работу, спит. Возвращает результат извлечения.
void wake_and_done()	Заканчивает работу очереди, пробуждая все потоки.
bool is_full_protected() const	Указывает, полна очередь или нет.
bool is_empty_and_done() const	Указывает, пуста и завершила ли свою работу очередь.
size_t get_size() const	Возвращает размер очереди.

class Call

Хранит информацию о входящем звонке.

Метод	**Описание**
Call(std::string phone_number)	Конструктор. Создает объект класса с указанным номером телефона, генерирует uuid, сохраняет дату

	звонка.
uuids::uuid get_id() const	Возвращает uuid звонка.
posix_time::ptime get_call_date() const	Возвращает дату звонка.
std::string get_phone_number() const	Возвращает номер телефона
bool is_timeout() const	Указывает, вышло ли время на обработку звонка.

class CallsQueue

Public-наследник класса TSQueue<Call>. Расширяет функционал для обработки повторных звонков.

Метод	**Описание**
CallsQueue(size_t queue_length)	Конструктор. Создает очередь звонков указанной длины.
size_t find_index(std::string phone_number)	Возвращает индекс звонка с указанным номером телефона.
Call get_call(size_t index)	Возвращает звонок с указанным индексом.
void erase(size_t index)	Убирает из очереди звонок с указанным индексом.

class Operator

Хранит информацию о работающем операторе.

Метод	**Описание**
void initialize()	Генерирует uuid.
uuids::uuid get_id() const	Возвращает uuid.
posix_time::ptime get_begin_date() const	Возвращает дату начала обработки звонка.
posix_time::ptime get_end_date() const	Возвращает дату конца обработки звонка
uint64_t get_working_time() const	Возвращает время обработки звонка (в секундах).
bool is_busy()	Указывает, занят ли оператор.
void set_busy()	Устанавливает оператору статус “Занят”. Генерирует время обработки. Запоминает дату начала работы.
void set_free()	Освобождает оператора. Сохраняет дату окончания работы.

class OperatorsManager

Распределяет звонки на операторов.

Метод	**Описание**
OperatorsManager(net::io_context& io_context, size_t operators_number, size_t queue_length)	Конструктор. Сохраняет ссылку на io_context, создает очереди звонков. Инициализирует заданное число операторов и заполняет ими очередь операторов.
~OperatorsManager()	Деструктор. Завершает работу очередей и работающего потока.
bool handle_call(Call call)	Кладет звонок в очередь звонков. В случае, если очередь звонков переполнена, или звонок повторный, делает соответствующую запись в журнал.
void start()	Запускает работающий поток, который ожидает появления данных в очередях, связывает звонок и оператора и спавнит корутину для последующей обработки и записи в журнал.
void stop()	Останавливает работающий поток.

class Controller

Обрабатывает HTTP-запросы.

Метод	**Описание**
Controller(net::io_context& io_context, OperatorsManager& operators_manager);	Конструктор. Сохраняет ссылку на io_context и OperatorsManager-а.
void start(std::string address, std::uint16_t port)	Спавнит слушающую указанный порт корутину, которая в свою очередь спавнит корутины для обработки приходящих запросов, отправления response, и направления звонка к OperatorsManager-у.
void stop()	Завершает работу controller-а.

class IOManager

Создает io_context, ожидает завершения работы программы.

Метод	**Описание**
IOManager(size_t workers_number)	Конструктор. Создает io_context. Инициализирует массив worker-ов заданной длины.
~IOManager()	Деструктор. Завершает работу io_context.
void start()	Запускает всех worker-ов.
void stop()	Завершает работу всех worker-ов.

net::io_context& get_context()	Возвращает ссылку на созданный io_context.

class Service

Запускает все службы. Ждет конца программы. Обрабатывает исключения.

Метод	**Описание**
Service(size_t workers_number, size_t operators_number, size_t queue_length);	Конструктор. Инициализирует IOManager, OperatorsManager и Controller с указанными параметрами.
void run(std::uint16_t port, std::string address)	Запускает IOManager, OperatorsManager и Controller. Ждет завершения работы программы. Обрабатывает исключения.

4. Тестирование

Классы CDR, Config, Call, Operator, IOManager и Service не нуждаются в тестировании, так как их действия либо тривиальны, либо шаблонны. Классы TSQueue и CallsQueue были успешно протестированы.

В ходе тестирования класса OperatorsManager возникла проблема тестирования асинхронных методов. Из-за спавна и отделения корутин от основной работы методов, тесты не дожидаются их завершения. Сделать тесты awaitable не представляется возможным, так как они определяются через макросы. По той же причине не удалось протестировать класс Controller.

TSQueue

Тест	**Описание**
SizeTest	Тестирует корректное возвращение размера очереди.
FullTest	Тестирует корректное возвращение логической переменной, указывающей полна очередь или нет.
EmptyAndDoneTest	Тестирует корректное завершение работы очереди.
PushPopTest	Тестирует корректное добавление и изъятие данных из очереди.
EmptyWaitTest	Тестирует работу пустой очереди.
FullWaitTest	Тестирует работу полной очереди.

```
matvey@zenbook15pro:~/Projects/ProteiCourseWork$ ./build/tests/ts_queue_test
[==========] Running 6 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 6 tests from TSQueueTest
[ RUN      ] TSQueueTest.SizeTest
[       OK ] TSQueueTest.SizeTest (0 ms)
[ RUN      ] TSQueueTest.FullTest
[       OK ] TSQueueTest.FullTest (0 ms)
[ RUN      ] TSQueueTest.EmptyAndDoneTest
[       OK ] TSQueueTest.EmptyAndDoneTest (0 ms)
[ RUN      ] TSQueueTest.PushPopTest
[       OK ] TSQueueTest.PushPopTest (0 ms)
[ RUN      ] TSQueueTest.EmptyWaitTest
[       OK ] TSQueueTest.EmptyWaitTest (1000 ms)
[ RUN      ] TSQueueTest.FullWaitTest
[       OK ] TSQueueTest.FullWaitTest (1000 ms)
[----------] 6 tests from TSQueueTest (2001 ms total)

[----------] Global test environment tear-down
[==========] 6 tests from 1 test suite ran. (2001 ms total)
[  PASSED  ] 6 tests.
```

CallsQueue

Тест	Описание
PushPopTest	Тестирует добавление и изъятие звонков из очереди.
FindIndexTest	Тестирует поиск звонка в очереди по номеру.
EraseTest	Тестирует удаление звонка из очереди.

```
matvey@zenbook15pro:~/Projects/ProteiCourseWork$ ./build/tests/calls_queue_test
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from CallsQueueTest
[ RUN      ] CallsQueueTest.PushPopTest
[       OK ] CallsQueueTest.PushPopTest (0 ms)
[ RUN      ] CallsQueueTest.FindIndexTest
[       OK ] CallsQueueTest.FindIndexTest (0 ms)
[ RUN      ] CallsQueueTest.EraseTest
[       OK ] CallsQueueTest.EraseTest (0 ms)
[----------] 3 tests from CallsQueueTest (0 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 3 tests.
```

OperatorsManager

Тест	Описание
HandleCallWithQueueFull	Тестирует обработку звонка с полной очередью.
HandleCallWithDuplicateHandlingOn	Тестирует обработку повторного звонка с включенным обработчиком.

HandleCallWithDuplicateHandlingoff	Тестирует обработку повторного звонка с отключенным обработчиком.

```
matvey@zenbook15pro:~/Projects/ProteiCourseWork$ ./build/tests/operators_manager_test
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from OperatorsManagerTest
[ RUN      ] OperatorsManagerTest.HandleCallWithQueueFull
[2023-12-02 01:42:05.578] [info] CDR is initialized
[2023-12-02 01:42:05.578] [info] operators manager starting...
[2023-12-02 01:42:05.578] [info] operators manager started. number of operators: 1. calls queue size: 10
[2023-12-02 01:42:05.589] [info] config is initialized
[2023-12-02 01:42:05.589] [info] CDR note is written. status: OVERLOAD
[2023-12-02 01:42:05.589] [info] CDR note is written. status: OVERLOAD
[2023-12-02 01:42:05.589] [info] CDR note is written. status: OVERLOAD
[2023-12-02 01:42:05.589] [info] CDR note is written. status: OVERLOAD
[2023-12-02 01:42:05.589] [info] CDR note is written. status: OVERLOAD
[2023-12-02 01:42:05.589] [info] operators manager stoped
[       OK ] OperatorsManagerTest.HandleCallWithQueueFull (11 ms)
[ RUN      ] OperatorsManagerTest.HandleCallWithDuplicateHandlingOn
[2023-12-02 01:42:05.589] [warning] CDR is already opened
[2023-12-02 01:42:05.589] [info] operators manager starting...
[2023-12-02 01:42:05.589] [info] operators manager started. number of operators: 1. calls queue size: 10
[2023-12-02 01:42:05.590] [info] config is initialized
[2023-12-02 01:42:05.590] [info] CDR note is written. status: CALL DUPLICATION
[2023-12-02 01:42:05.590] [info] operators manager stoped
[       OK ] OperatorsManagerTest.HandleCallWithDuplicateHandlingOn (0 ms)
[ RUN      ] OperatorsManagerTest.HandleCallWithDuplicateHandlingoff
[2023-12-02 01:42:05.590] [warning] CDR is already opened
[2023-12-02 01:42:05.590] [info] operators manager starting...
[2023-12-02 01:42:05.590] [info] operators manager started. number of operators: 1. calls queue size: 10
[2023-12-02 01:42:05.590] [info] config is initialized
[2023-12-02 01:42:05.590] [info] CDR note is written. status: ALREADY IN QUEUE
[2023-12-02 01:42:05.590] [info] operators manager stoped
[       OK ] OperatorsManagerTest.HandleCallWithDuplicateHandlingoff (0 ms)
[----------] 3 tests from OperatorsManagerTest (12 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (12 ms total)
[  PASSED  ] 3 tests.
Segmentation fault (core dumped)
```

Тесты проходятся успешно, однако по завершении по указанным выше причинам падает Segmentation Fault. По этим же причинам не удалось протестировать остальной функционал класса.

5. Демонастрация

Конфигурация

```
{
    "address" : "127.0.0.1",
    "port" : 8000,
    "workers_number" : 5,
    "queue_length" : 10,
    "operators_number" : 2,
    "handling_min_time" : 5,
    "handling_max_time" : 15,
    "is_call_dupplicaton_handler_on" : true
}
```

Запуск

```
matvey@zenbook15pro:~/Projects/ProteiCourseWork/build$ ./src/ProteiCourseWork ./src/config.json
[2023-12-02 01:46:40.425] [info] config is initialized
[2023-12-02 01:46:40.425] [info] CDR is initialized
[2023-12-02 01:46:40.425] [info] service is initialized
[2023-12-02 01:46:40.425] [info] service running...
[2023-12-02 01:46:40.425] [info] operators manager starting...
[2023-12-02 01:46:40.426] [info] controller starting...
[2023-12-02 01:46:40.426] [info] I/O manager starting...
[2023-12-02 01:46:40.426] [info] worker started: 4271422181507087194
[2023-12-02 01:46:40.426] [info] operators manager started. number of operators: 2. calls queue size: 10
[2023-12-02 01:46:40.426] [info] controller is listening: 127.0.0.1:8000
[2023-12-02 01:46:40.426] [info] worker started: 5881448337415220859
[2023-12-02 01:46:40.426] [info] worker started: 5777645207055158873
[2023-12-02 01:46:40.426] [info] worker started: 11232804704249288046
[2023-12-02 01:46:40.427] [info] I/O manager started. workers number: 5
[2023-12-02 01:46:40.427] [info] worker started: 16359176917212970279
```

Обработка запроса и запись в журнал

```
[2023-12-02 01:47:28.386] [info] acception is being handled...
[2023-12-02 01:47:28.386] [info] response created. call id: d2251606-3225-4951-a099-11e129f4aaff
[2023-12-02 01:47:28.386] [info] request handled
[2023-12-02 01:47:28.386] [info] operator 05c27a1a-04cc-4777-abe9-6652cf3c82d7 is working...
[2023-12-02 01:47:28.386] [info] acception is handled
[2023-12-02 01:47:36.387] [info] operator 05c27a1a-04cc-4777-abe9-6652cf3c82d7 is free...
[2023-12-02 01:47:36.387] [info] CDR note is written. status: OK
```

```
build > ≡ CDR.txt
  1  2023-12-02T01:47:28;d2251606-3225-4951-a099-11e129f4aaff;+79995264052;2023-12-02T01:47:36;OK;2023-12-02T01:47:28;05c27a1a-04cc-4777-abe9-6652cf3c82d7;00:00:08
```

Завершение

```
^C[2023-12-02 01:50:00.887] [info] I/O manager stopped
[2023-12-02 01:50:00.887] [info] worker stopped: 4271422181507087194
[2023-12-02 01:50:00.887] [info] worker stopped: 5777645207055158873
[2023-12-02 01:50:00.887] [info] worker stopped: 11232804704249288046
[2023-12-02 01:50:00.887] [info] service is stopping...
[2023-12-02 01:50:00.887] [info] worker stopped: 16359176917212970279
[2023-12-02 01:50:00.887] [info] worker stopped: 5881448337415220859
[2023-12-02 01:50:00.888] [info] operators manager stoped
[2023-12-02 01:50:00.888] [info] controller stopped
[2023-12-02 01:50:00.888] [info] service finished successfully
[2023-12-02 01:50:00.888] [info] I/O manager stopped
[2023-12-02 01:50:00.888] [info] CDR is closed
```