**О программе**

Программа написана в процессе выполнения проектного задания IT-школы Протей. Программа является сервером, который осуществляет прием вызовов в форме http-запросов, формирует ответы, распределяет вызовы на операторов и ведет журнал с записями о каждом вызове. Некоторые параметры программы могут настраиваться, для этого используется файл конфигурации.

Для создания программы использован фреймворк Qt (Qt Creator 6.0.2). Используются классы, предоставляемые Qt, например, QDateTime, QTimer, QThread, QMutex и другие.

Для реализации сервера, принимающего запросы через сеть, используется библиотека boost (boost/beast, boost/asio), для ведения лога используется boost/log. Лог выводится в файл log.txt.

Для проведения unit-тестов используется библиотека googletest (тесты представлены в виде отдельного зависимого проекта, расположенного в папке Test).

Для сборки проектов используется CMake. Код программы написан только на С++.

# Структура программы

Программа состоит из 3 основных частей.

1. Сервер. Сервер принимает вызов (http-запрос) и возвращает ответ. В ответе содержится CallID, если запрос принят и помещен в очередь. Если очередь запросов заполнена, в очереди уже есть такой номер (дублирование номера) или номер некорректный, в ответе будет соответствующее сообщение.
2. Контроллер очереди. Контроллер помещает номер и CallID в очередь, распределяет заявки из очереди на операторов, удаляет заявки с истекшим временем ожидания. В этой же части происходит эмуляция ответа оператора на вызов.
3. CDR writer. Получает информацию о вызовах и изменении их состояния, систематизирует её и выводит в файл CDR.txt.

Каждая из этих частей работает в своем потоке.

# Конфигурация и запуск программы.

В приложении есть набор значений, которые могут быть настроены. Файл конфигурации config.json содержит значения, перечисленные в таб.1.

*Таблица 1*

|  |  |  |
| --- | --- | --- |
| Ключи | Назначение | Значение по умолчанию |
| queue size | Размер очереди для вызовов | 20 |
| operators number | Количество операторов | 5 |
| wait min time sec | Нижний предел времени ожидания (секунды) | 100 |
| wait max time sec | Верхний предел времени ожидания (секунды) | 1000 |
| operator busy min time sec | Нижний предел времени обработки вызова оператором (секунды) | 30 |
| operator busy max time sec | Верхний предел времени обработки вызова оператором (секунды) | 120 |

Запуск приложения осуществляется через терминал Linux. Во время запуска программы можно ввести путь к файлу конфигурации. Пример:

./CallCenter\_Server /home/user/example/config.json

Если не ввести путь к файлу, программа использует путь по умолчанию – попытается найти файл с конфигурацией в том же каталоге, где находится исполняемый файл.

Если путь к файлу введен неверно, и файл не будет обнаружен, программа установит значения по умолчанию и выведет сообщение об этом в терминал.

# Классы программы

В программе созданы следующие классы:

*Таблица 2*

|  |  |
| --- | --- |
| **Класс** | **Назначение** |
| Configjson | Считывает файл конфигурации (config.json). Проверяет значения; устанавливает значения по умолчанию в случае, если config.json отсутствует или содержит некорректные данные. |
| Server | Принимает запросы http, извлекает из них номер вызова, назначает callID. Отправляет ответ: callID или вид ошибки (overload, call duplication, incorrect number) |
| Serverworker | Контролирует очередь – проверяет случаи overload и call duplication. Назначает номер из очереди оператору; убирает из очереди вызовы с превышением времени ожидания (timeout). |
| Callprocessing | Класс оператора. Эмулирует ответ оператора с помощью таймера. Меняет состояние оператора (занят/свободен). |
| Caller | Класс абонента. Содержит номер телефона, callID, информацию о времени ожидания (для эмуляции timeout). |
| CDRWorker | Собирает информацию от остальных классов программы для ведения CDR. Группирует информацию, относящуюся к одному и тому же вызову. Пишет CDR в файл CDR.txt |
| WorkerStatus | Класс для управления ответом сервера – содержит только перечисление (enum) статусов (OK, DEFAULT, OVERLOAD, DUPLICATION ). |

* 1. **Configjson.**

Класс предназначен для чтения файла конфигурации и передачи настроек другим классам. При чтении файла происходят проверки – существует ли файл, нет ли ошибок при парсинге JSON.

При чтении значений происходит проверка условий:

1. Должны присутствовать все перечисленные в таб.1 ключи.
2. Все значения должны быть целыми положительными числами.
3. Значения верхнего (max) и нижнего (min) пределов времени должны иметь правильное соотношение (минимальное значение меньше максимального).
4. Минимальное количество операторов равно 1, минимальная длина очереди равна 1.

Нарушение любого из этих условий приведет к установке параметров по умолчанию для всех значений.

* 1. **Server**

Сервер принимает post-запрос http с номером. Номер должен состоять из 10 цифр, допускается символ «+» в начале номера. Неправильное количество цифр или присутствие любых других символов (букв, знаков) вызывает ответ incorrect number. Работа с данным запросом завершается.

Сервер формирует идентификатор CallID из номера абонента и времени поступления вызова (числа миллисекунд с начала дня до момента поступления вызова). Используется операция XOR (побитовое исключающее ИЛИ) для этих двух величин. CallID состоит из 10 цифр, как и номер телефона.

Сервер отправляет номер и ID контроллеру очереди и получает от него ответ в виде статуса. В зависимости от статуса сервер формирует ответ:

*Таблица 2*

|  |  |
| --- | --- |
| **Статус** | **Ответ** |
| OVERLOAD | "error: server is overloaded" |
| DUPLICATION | "error: call duplication - already in queue" |
| OK | <CallID> |

После этого работа сервера с запросом завершается.

* 1. **Serverworker**

Контроллер очереди содержит две основные функции.

WorkerStatus checkQueue( Caller &currentCaller)

Эта функция вызывается, если получен запрос с корректным номером. Контроллер очереди проверяет размер очереди и наличие совпадающего номера в ней. Номер добавится в очередь, если она не переполнена и нет дублирования номера. Функция вернет статус вызова (табл.2 выше).

void ServerWorker::maintainQueue()

Функция вызывается периодически по таймеру. Контроллер сперва проверяет, нет ли заявок с истекшим временем ожидания (timeout), затем распределяет заявки из очереди на свободных операторов (если в очереди есть заявки и есть свободные операторы).

* 1. **Callprocessing**

Класс оператора. При запуске программы создается вектор объектов этого класса. Каждый оператор имеет свой порядковый номер (начиная с 0), свой таймер, и получает из файла конфигурации пределы времени для обслуживания вызова. Оператор может быть в состоянии «свободен» или «занят».

При распределении вызова оператор получает номер телефона и callID этого вызова и переходит в состояние «занят». Генерируется случайное время в пределах, указанных в конфигурации. Оператор запускает свой таймер на это время. Когда таймер завершается, оператор переходит в состояние «свободен».

* 1. **Caller**

Класс абонента. Объект такого класса создается при поступлении вызова с корректным номером и добавляется в очередь, если нет перегрузки или дублирования вызовов. У абонента есть номер телефона, callID и время поступления вызова.

Для каждого абонента генерируется случайное время отбоя в пределах, указанных в конфигурации. Абоненты удаляются из очереди либо при передаче вызова оператору, либо при превышении времени отбоя.

* 1. **CDRWorker**

Класс для ведения журнала вызовов. Содержит структуру record с параметрами отдельного вызова. В классе также есть вектор journal, состоящий из структур record – он представляет собой журнал, в котором хранятся данные о вызовах внутри программы, до вывода в файл CDR.txt. Если запись отправлена на вывод в файл, она удаляется из журнала.

*Таблица 3*

|  |  |
| --- | --- |
| **Структура record** | |
| **Поле** | **Назначение** |
| startCallDT | дата/время поступления вызова |
| callID | идентификатор вызова |
| phoneNumber | номер телефона абонента |
| finCallDT | дата/время окончания вызова |
| status | статус (принят, перегрузка, превышено время ожидания, дублирование) |
| answDT | дата/время ответа оператора |
| operNum | порядковый номер оператора |
| callDuration | длительность разговора (разность времени ответа оператора и времени окончания вызова) |

Статусы вызова заданы перечислением callStatus: CALL\_OK, TIMEOUT, OVERLOAD, CALL\_DUPLICATION, NOT\_FINISHED. Это перечисление **не** связано с перечислением WorkerStatus. Оно предназначено только для класса CDRWorker.

Класс CDRWorker имеет набор слотов – функций, вызываемых сигналами из разных частей программы. Сигналы поступают, когда происходит одно из событий:

1. Новый вызов добавлен в очередь;
2. Оператор начал ответ на вызов из очереди;
3. Оператор закончил обслуживание вызова;
4. Истекло время ожидания вызова в очереди (timeout)
5. Вызов не получилось добавить в очередь из-за переполнения (overload)
6. Вызов не получилось добавить в очередь из-за дублирования номера.

Сигнал содержит CallID вызова, с которым произошло какое-то из перечисленных событий. Сигнал может содержать и другую информацию, например, время события, номер абонента. В зависимости от события в журнале создается новая запись или дополняется существующая. Если информации о вызове достаточно, формируется строка, которая выводится в файл CDR.txt средствами boost log. Таблица ниже содержит описание слотов CDRWorker.

*Таблица 4*

|  |  |
| --- | --- |
| Функция | Действия |
| recInCall | Создать новую запись в журнале;  Добавить в запись:   * время поступления вызова * номер телефона абонента * CallID абонента * установить статус NOT\_FINISHED |
| recAnswerCall | Найти запись с указанным CallID;  Добавить в запись:   * время ответа оператора * порядковый номер оператора |
| recFinishAnsweredCall | Найти запись с указанным CallID;  Добавить в запись:   * время окончания ответа * рассчитанную продолжительность разговора * установить статус CALL\_OK   Вывести запись в файл;  Удалить запись из журнала |
| recCallOverload | Создать новую запись в журнале;  Добавить в запись:   * время поступления вызова * номер телефона абонента * CallID абонента * установить статус OVERLOAD * заполнить остальные поля пустыми значениями   Вывести запись в файл;  Удалить запись из журнала |
| recTimeoutedCall | Найти запись с указанным CallID;  Добавить в запись:   * время отбоя * установить статус TIMEOUT   Вывести запись в файл;  Удалить запись из журнала |
| recCallDuplication | Создать новую запись в журнале;  Добавить в запись:   * время поступления вызова * номер телефона абонента * CallID абонента * установить статус CALL\_DUPLICATION * заполнить остальные поля пустыми значениями   Вывести запись в файл;  Удалить запись из журнала. |

Если найти запись с указанным CallID не получается, ничего не происходит.

# Слоты и сигналы программы

Qt предоставляет механизм слотов и сигналов, который позволяет вызвать функцию, например, при изменении состояния объекта. На рисунке ниже приведена диаграмма слотов и сигналов, которые используют классы программы. В основном объекты классов сигнализируют друг другу об изменении состояния вызова, например: вызов принят, вызов назначен оператору, вызов окончен, вызов является дублирующим и т.д.

Эмуляция оператора реализуется с помощью таймера. Таймер запускается один раз, после окончания таймера (сигнал timeout) оператор считается освободившимся.

Управление очередью (проверка вызовов с истекшим временем ожидания и распределение вызова на свободных операторов, если они есть) также происходит с помощью функции, которая запускается периодически по таймеру.

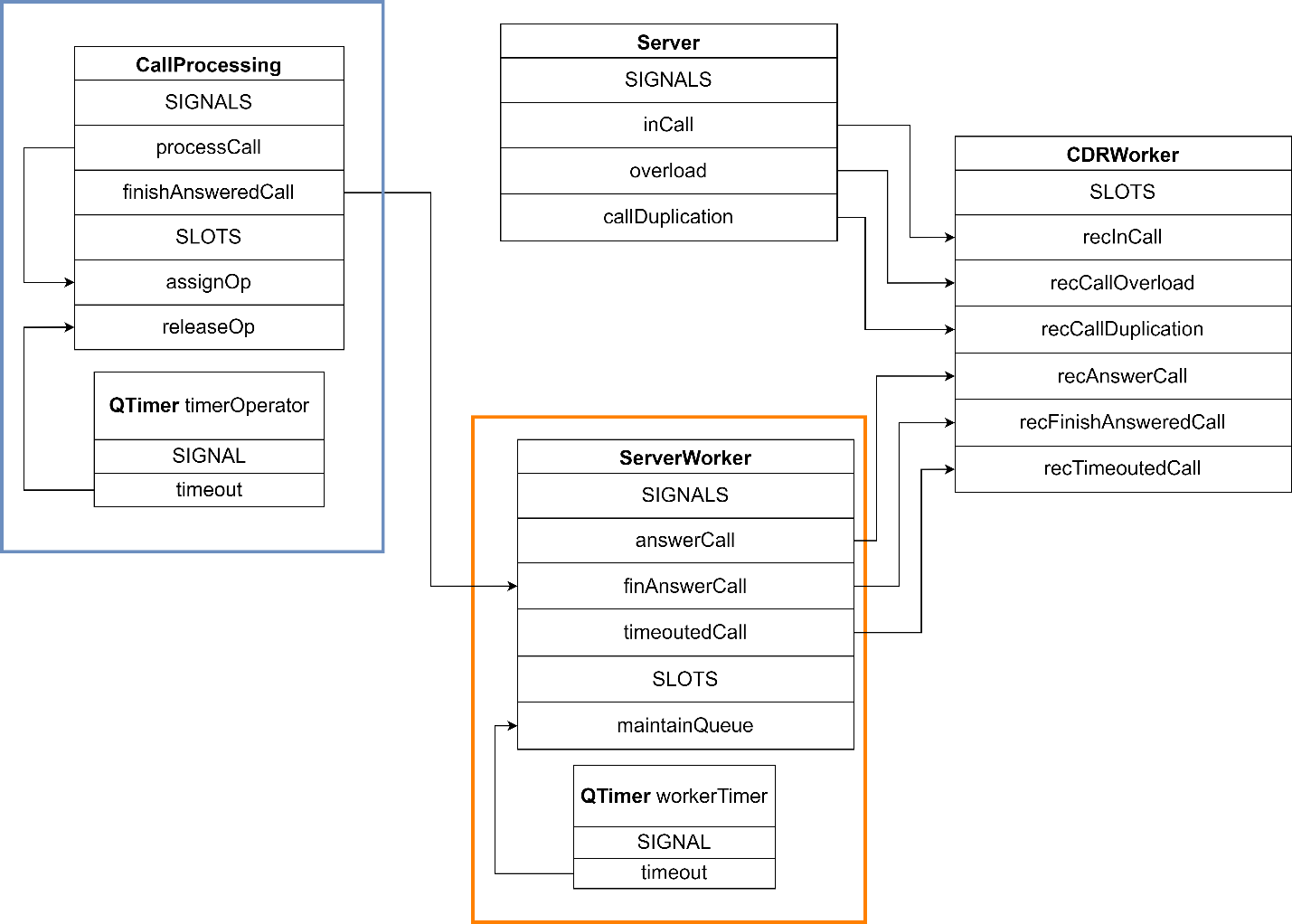


Рис. 1. Диаграмма слотов и сигналов

На диаграмме не приведены три слота startWorker, startCDR, runServer. Эти слоты вызываются только один раз - во время запуска сервера.

# Примеры отправки запросов

Для проверки работоспособности сервера использовалась утилита curl и сайт <https://reqbin.com/post-online>. Примеры отправки запросов и вывода ответов приведены ниже.

Дли использования утилиты curl можно ввести строку, например:

curl -X POST -d "7812334232" <http://localhost:8080>

Также можно использовать bash-скрипт для отправки нескольких запросов, например:

|  |
| --- |
| #!/bin/bashfor i in {1..5}do number=$((1000000000 + $RANDOM % 9000000000)) echo "Отправка запроса с номером ${number}" curl -X POST -d "${number}" http://localhost:8080 echo ""  done |

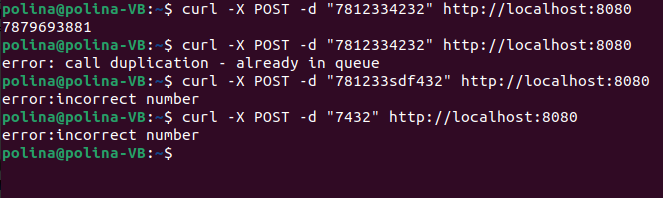


Рис. 3. Примеры ответа сервера на запросы

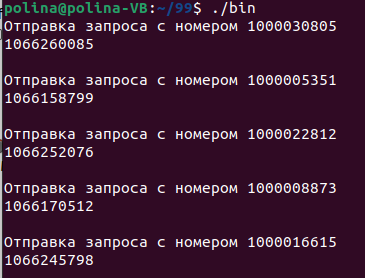


Рис. 3. Примеры ответа сервера на запросы