МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Московский Авиационный Институт

(Национальный исследовательский университет)

Институт №8

«Компьютерные науки и прикладная математика»

Кафедра 806

«Вычислительная математика и программирование»

Курсовой проект по дисциплине «Фундаментальные алгоритмы»

Тема: «Разработка алгоритмов системы хранения и управления данными на основе динамических структур данных»

Студент: Суляева Алина Игоревна

Группа: М8О-211Б-21

Преподаватель: Ирбитский И.С.

Оценка:

Дата:

# Введение

**Задание курсового проекта:**

На языке программирования C++ (стандарт C++14 и выше) реализуйте приложение, позволяющее выполнять операции над коллекциями данных заданных типов (типы обслуживаемых объектов данных определяются вариантом) и контекстами их хранения (коллекциями данных). Коллекция данных описывается набором строковых параметров (набор параметров однозначно идентифицирует коллекцию данных):

● название пула схем данных, хранящего схемы данных;

● название схемы данных, хранящей коллекции данных;

● название коллекции данных.

Коллекция данных представляет собой ассоциативный контейнер (конкретная реализация определяется вариантом), в котором каждый объект данных соответствует некоторому уникальному ключу. Для ассоциативного контейнера необходимо вынести интерфейсную часть (в виде абстрактного класса C++) и реализовать этот интерфейс. Взаимодействие с коллекцией объектов происходит посредством выполнения одной из операций над ней:

● добавление новой записи по ключу;

● чтение записи по ее ключу;

● чтение набора записей с ключами из диапазона [𝑚𝑖𝑛𝑏𝑜𝑢𝑛𝑑... 𝑚𝑎𝑥𝑏𝑜𝑢𝑛𝑑]; ● обновление данных для записи по ключу;

● удаление существующей записи по ключу.

Во время работы приложения возможно выполнение также следующих операций:

● добавление/удаление пулов данных;

● добавление/удаление схем данных для заданного пула данных;

● добавление/удаление коллекций данных для заданной схемы данных заданного пула данных.

Поток команд, выполняемых в рамках работы приложения, поступает из файла, путь к которому подаётся в качестве аргумента командной строки. Формат команд в файле определите самостоятельно.

**Дополнительные задания, реализованные в этой курсовом проекте:**

1. Реализуйте интерактивный диалог с пользователем. Пользователь при этом может вводить конкретные команды (формат ввода определите самостоятельно) и подавать на вход файлы с потоком команд.

2. Обеспечьте хранение объектов строк, размещенных в объектах данных, на основе структурного паттерна проектирования “Приспособленец”. Дублирования объектов строк для разных объектов (независимо от контекста хранения) при этом запрещены. Доступ к строковому пулу обеспечьте на основе порождающего паттерна проектирования “Одиночка”.

3. Реализуйте функционал приложения в виде сервера, запросы на который поступают из клиентских приложений. При этом взаимодействие клиентских приложений с серверным должно быть реализовано посредством средств межпроцессного взаимодействия (IPC). Используемые средства IPC и операционная система определяются вариантом.

4. Реализуйте комплекс серверных приложений (далее: кластер), одно из которых (далее: entrypoint) обрабатывает входящие пользовательские запросы и делегирует их остальным серверам (далее: storage), а остальные сервера хранят данные. Распределение данных по storage серверам должно быть примерно равномерным по памяти в произвольный момент времени.

5. Реализуйте серверное приложение, собирающее логи клиентской (и, если есть, серверной) части приложения в файловые потоки вывода. Конфигурирование серверного логгера обеспечьте на основе файла со структурой JSON.

**Вариант курсового проекта 35:**

Тип данных:

3. Информация о прохождении контеста соискателем (id соискателя, ФИО соискателя (раздельные поля), дата рождения соискателя, ссылка на резюме соискателя, id закрепленного за соискателем HR-менеджера, id контеста, ЯП на котором реализуются задачи контеста, кол-во задач в контесте, кол-во решённых задач в контесте, было ли обнаружено списывание с микронаушником).

Контейнер: B+ дерево

IPC: Windows shared memory + Windows semaphores

# Описание реализованного приложения

В результате выполнения курсового проекта было создано многофункциональное приложение, которое позволяет формировать, хранить, добавлять, удалять и просматривать информацию о прохождении контеста соискателем. Данное приложение используется для структурирования и организации данных, а также для быстрого поиска нужных записей.

Созданное приложение, основанное на классе ClientProcessor, имеет возможность взаимодействия с пользователем через интерфейс и чтения запросов из текстового файла в формате CSV. При запуске приложения, поток команд из указанного файла будет выполнен, и пользователь получит вывод результата запроса в консоли. Класс ClientProcessor также позволяет пользователю отправлять запросы через консоль и получать ответы на них.

После получения запроса, ClientProcessor отправляет его на сервер, который обрабатывает запрос и возвращает ответ с определенным статусом. Для обработки запроса сервер отправляет запрос на другие серверы, которые являются хранилищами данных. Выделенный сервер используется для регистрации событий на всех серверах.

Решение включает несколько классов, которые обеспечивают взаимодействие между клиентом и сервером, обработку запросов, хранение данных и регистрацию событий.

Класс ClientProcessor:

– имя разделяемой памяти (shared\_memory\_name) - имя разделяемой памяти, используемой для взаимодействия с сервером;

– имя мьютекса (mutex\_name) - имя мьютекса, используемого для установления соединения с сервером;

– соединение с сервером (connection) - объект класса Connection, используемый для взаимодействия с сервером.

Основные методы класса:

– функция чтения команд из файла (read\_commands\_from\_file) - принимает путь к файлу в формате CSV с определенной структурой команд, читает команды из файла и отправляет их на сервер с помощью объекта Connection, выводит результат выполнения каждой команды в консоль;

– функция создания запросов через консоль (interactive\_mode) - запрашивает у пользователя данные для создания запроса, отправляет запрос на сервер с помощью объекта Connection и выводит результат выполнения в консоль;

– функция отправки запроса на сервер (send\_request) - принимает запрос в виде словаря, отправляет его на сервер с помощью объекта Connection и возвращает результат выполнения запроса;

– функция обработки ответа от сервера (process\_response) - принимает ответ от сервера в виде словаря, проверяет статус ответа и возвращает результат выполнения запроса или сообщение об ошибке.

Класс ClientProcessor является важной частью приложения и обеспечивает удобный интерфейс для взаимодействия клиента с сервером.

Класс ClientProcessor имеет следующие основные поля:

– thisStatusCode: константа, хранящая код состояния процессора;

– connection: указатель на объект класса Connection, представляющий соединение с сервером;

– connectionName: имя соединения;

– logger: ссылка на объект класса ServerLogger, используемый для ведения журнала.

Конструктор класса принимает следующие аргументы:

– statusCode: код состояния процессора;

– memNameForConnect: имя разделяемой памяти, используемой для соединения;

– mutexNameForConnect: имя мьютекса, используемого для соединения;

– serverLogger: ссылка на объект класса ServerLogger, используемый для ведения журнала.

Класс ClientProcessor имеет следующие основные поля:

– thisStatusCode: константа, хранящая код состояния процессора;

– connection: указатель на объект класса Connection, представляющий соединение с сервером;

– connectionName: имя соединения;

– logger: ссылка на объект класса ServerLogger, используемый для ведения журнала.

Конструктор класса принимает следующие аргументы:

– statusCode: код состояния процессора;

– memNameForConnect: имя разделяемой памяти, используемой для соединения;

– mutexNameForConnect: имя мьютекса, используемого для соединения;

– serverLogger: ссылка на объект класса ServerLogger, используемый для ведения журнала.

Методы add, get, contains и remove принимают параметры для определения базы данных, схемы, таблицы и значения контеста для выполнения операции. Они отправляют запрос на сервер и ожидают ответа. Ответ анализируется, и возвращается соответствующий результат операции.

Методы removeDatabase, removeSchema и removeTable выполняют удаление соответствующих элементов (базы данных, схемы, таблицы) с помощью запроса на сервер и анализа полученного ответа.

Метод log записывает сообщение в журнал с указанным уровнем серьезности.

Метод process является пустой реализацией виртуального метода process базового класса Processor и может быть переопределен в дочерних классах.

Класс также содержит приватные методы waitResponse, generateRandomContestInfoString, readContestInfoFromString, readContestInfoFromCin и readIntFromCin, которые выполняют вспомогательные операции, такие как ожидание ответа от сервера, генерация случайных данных контеста и чтение данных контеста из строки или стандартного ввода.

Наконец, класс содержит метод interactiveMenu, который является главным методом класса ClientProcessor. Он запускает интерактивный режим взаимодействия с пользователем, где пользователь может выбрать различные операции для выполнения на сервере.

Внутри метода interactiveMenu создается бесконечный цикл, в котором пользователю предлагается выбрать операцию. Для выбора операции пользователь должен ввести соответствующий символ. Доступные операции включают добавление, получение, проверку на существование и удаление контестов, а также удаление баз данных, схем и таблиц.

После выбора операции пользователь может потребоваться ввести дополнительные параметры, такие как имя базы данных, схемы, таблицы или значения контеста. Затем метод вызывает соответствующий метод класса ClientProcessor для выполнения выбранной операции с указанными параметрами.

Результат операции отображается пользователю на экране. Если операция выполнена успешно, отображается соответствующее сообщение. В случае ошибки отображается сообщение об ошибке.

Метод interactiveMenu также обрабатывает исключения, которые могут возникнуть во время взаимодействия с сервером или из-за некорректного ввода пользователя.

Класс ClientProcessor содержит поля thisStatusCode, connection, connectionName и logger, а также конструктор, который принимает аргументы для установки соединения и журнала.

Класс также содержит методы для выполнения операций с базами данных, схемами, таблицами и контестами, а также методы для удаления элементов базы данных, схемы и таблицы. Есть также методы для записи сообщений в журнал и обработки пользовательского ввода в интерактивном режиме.

Внутри метода interactiveMenu создается бесконечный цикл, в котором пользователю предлагается выбрать операцию. После выбора операции пользователь может потребоваться ввести дополнительные параметры, такие как имя базы данных, схемы, таблицы или значения контеста. Затем метод вызывает соответствующий метод класса ClientProcessor для выполнения выбранной операции с указанными параметрами.

Метод interactiveMenu также обрабатывает исключения, которые могут возникнуть во время взаимодействия с сервером или из-за некорректного ввода пользователя.

Класс Candidate содержит следующие атрибуты:

– candidate\_id: идентификатор кандидата;

– last\_name: фамилия кандидата (хранится в StringPool);

– first\_name: имя кандидата (хранится в StringPool);

– patronymic: отчество кандидата (хранится в StringPool);

– birth\_date: дата рождения кандидата (хранится в StringPool);

– resume\_link: ссылка на резюме кандидата (хранится в StringPool);

– hr\_manager\_id: идентификатор HR-менеджера;

– contest\_id: идентификатор контеста;

– programming\_language: язык программирования (хранится в StringPool);

– num\_tasks: количество задач в контесте;

– solved\_tasks: количество решенных задач;

– cheating\_detected: флаг, указывающий на обнаружение попытки мошенничества.

Класс Candidate имеет следующие атрибуты:

– candidate\_id: идентификатор кандидата;

– last\_name: фамилия кандидата (хранится в StringPool);

– first\_name: имя кандидата (хранится в StringPool);

– patronymic: отчество кандидата (хранится в StringPool) ;

– birth\_date: дата рождения кандидата (хранится в StringPool);

– resume\_link: ссылка на резюме кандидата (хранится в StringPool);

– hr\_manager\_id: идентификатор менеджера по персоналу;

– contest\_id: идентификатор контеста;

– programming\_language: язык программирования (хранится в StringPool);

– num\_tasks: количество задач в контесте;

– solved\_tasks: количество решенных задач;

– cheating\_detected: флаг, указывающий на обнаружение попытки мошенничества.

Методы класса:

Конструктор ContestInfo принимает значения всех атрибутов и инициализирует их.

Конструктор копирования ContestInfo создает копию объекта.

Статический метод get\_obj\_for\_search возвращает объект ContestInfo с заполненными значениями candidate\_id и contest\_id. Используется для поиска контеста.

Метод hashcode вычисляет хэш-код объекта на основе значений candidate\_id и contest\_id.

Оператор сравнения operator== сравнивает два объекта ContestInfo на равенство на основе значений candidate\_id и contest\_id.

Оператор присваивания operator= присваивает один объект ContestInfo другому.

Метод serialize сериализует объект в строку.

Статический метод deserialize десериализует строку и создает объект ContestInfo.

Несколько геттеров используются для получения значений атрибутов.

Метод print выводит информацию о контесте в консоль.

Класс также использует классы boost::archive::text\_oarchive и boost::archive::text\_iarchive для сериализации и десериализации объектов.

Класс SharedObject представляет объект, передаваемый через разделяемую память. Он содержит данные запроса или ответа, а также информацию о том, был ли запрос обработан.

Как и класс ContestInfo, SharedObject имеет методы для сериализации и десериализации.

В конструкторе класса SharedObject принимается объект, который наследуется от класса Serializable, и сохраняется. Этот класс позволяет обмениваться данными через разделяемую память.

Класс SharedObject представляет общий объект для передачи данных между клиентами, сервером и хранилищами в системе. Он наследуется от класса Serializable, что позволяет сериализовать и десериализовать объект для передачи по сети.

Класс содержит следующие поля:

– request\_response\_code - символ, определяющий код запроса или ответа;

– status\_code - символ, обозначающий обработку данных;

– data - строка, содержащая данные объекта. Может быть "null", если данных нет.

Класс также предоставляет методы для сериализации и десериализации объекта, получения кода состояния, кода запроса/ответа и данных объекта, а также установки кода состояния.

Методы serialize и deserialize используются для преобразования объекта в строку и обратно. Метод serialize объединяет все данные объекта в строку, используя разделители и форматирование. Метод deserialize извлекает данные из строки и создает новый объект SharedObject на основе этих данных.

Методы getStatusCode и getRequestResponseCode возвращают код состояния и код запроса/ответа соответственно.

Метод getData возвращает данные объекта в виде строки, обернутой в std::optional. Если данные равны "null", метод возвращает std::nullopt.

Метод setStatusCode позволяет установить новый код состояния объекта.

Методы print и getPrint используются для вывода информации о объекте в консоль или в виде строки.

Класс SharedObject является важным компонентом системы, позволяющим обмениваться данными между различными компонентами.

Класс MemoryConnection отвечает за создание и отправку сообщений через разделяемую память. Он обеспечивает удобную работу с межпроцессорным взаимодействием, используя библиотеку Boost.interprocess, которая позволяет кроссплатформенно работать с приложением.

Класс MemoryConnection является наследником класса Connection и представляет собой соединение с использованием разделяемой памяти (shared\_memory) в операционной системе.

Описание полей класса:

mreg - указатель на объект класса mapped\_region, который представляет сегмент разделяемой памяти;

is\_server - флаг, указывающий, является ли текущий объект соединения сервером.

Если объект соединения является сервером, то происходит удаление существующего сегмента разделяемой памяти с указанным именем (если он существует) и создание нового сегмента разделяемой памяти с указанным именем. Затем создается объект mapped\_region, связанный с этим сегментом разделяемой памяти.

Если объект соединения является клиентом, то происходит открытие существующего сегмента разделяемой памяти с указанным именем. Затем создается объект mapped\_region, связанный с этим сегментом разделяемой памяти.

Деструктор класса освобождает ресурсы, связанные с объектом соединения. Если объект соединения является сервером, то происходит удаление сегмента разделяемой памяти с указанным именем. Затем освобождается память, выделенная под объект mapped\_region.

Обратите внимание, что в деструкторе используется оператор delete для освобождения памяти, выделенной оператором new, поскольку объект mreg создается динамически.

Описание методов класса:

Метод receiveMessage переопределяет виртуальный метод из базового класса Connection. Он возвращает указатель на адрес начала сегмента разделяемой памяти, который используется для получения сообщения.

Метод sendMessage переопределяет виртуальный метод из базового класса Connection. Он принимает объект типа Serializable и отправляет его содержимое в сегмент разделяемой памяти. Первый байт в сегменте используется для хранения статуса, а остальные байты используются для хранения сериализованных данных.

Все запросы, поступающие от клиентов, обрабатываются классом ServerProcessor. Когда клиент запрашивает подключение, сервер создает соединение и добавляет клиента в список.

Сервер регулярно проверяет разделяемую память на наличие запросов от клиентов. Если есть запрос, который не требует обращения к базе данных, сервер отвечает на него. Если требуется обращение к базе данных, сервер определяет соответствующие серверы, на которых расположена база данных, и отправляет им запрос. После получения ответа сервер отправляет ответ клиенту или ожидает необходимое количество ответов.

В конструкторе класса ServerProcessor требуется передать имена разделяемой памяти серверов хранилищ для установления связи с ними.

Класс ServerProcessor является наследником класса Processor и представляет собой серверный процессор для обработки запросов клиентов.

Описание полей класса:

– storages - вектор объектов типа Storage, представляющих хранилища данных;

– clients - вектор указателей на объекты типа Connection, представляющих клиентские соединения.

Сервер хранилища базы данных представлен классом StorageProcessor. Он обрабатывает запросы, полученные от сервера, и отправляет ответы.

В конструктор класса StorageProcessor передается имя разделяемой памяти для установления связи с сервером.

Класс StorageProcessor является наследником класса Processor и представляет собой процессор для обработки запросов к хранилищу данных.

Описание полей класса:

– db - объект типа BPlusTreeMap, представляющий базу данных;

– this\_status\_code - статусный код процессора;

– connection - указатель на объект типа Connection,

Класс ServerLogger предназначен для отправки сообщений на сервер журналов.

Описание полей класса:

– connection - указатель на объект типа Connection, представляющий соединение с сервером журналов;

– queue - очередь сообщений для отправки на сервер журналов;

– mutex - мьютекс для синхронизации доступа к очереди сообщений.

Оба класса являются важными компонентами системы для регистрации и отслеживания событий.

Класс BPlusTreeMap описывает реализацию B+ дерева для хранения пар ключ-значение. Внутри класса определены структуры Entry и Node, используемые для хранения данных в дереве.

Основные члены класса:

– degree: переменная, хранящая степень дерева;

– leafCapacity: переменная, определяющая максимальное количество элементов в листе дерева;

– minLeafSize, maxChildCount, maxKeysCount, minChildCount, minKeysCount: переменные, вычисляющие минимальное и максимальное количество дочерних узлов и ключей в узле на основе степени дерева и leafCapacity;

– alloc: умный указатель на объект Memory, используемый для выделения и освобождения памяти;

– compare: функция сравнения двух ключей типа K;

– size\_: переменная, хранящая текущее количество элементов в дереве;

– root: указатель на корневой узел дерева.

Класс также определяет приватные методы и функции-члены, такие как createEntry, destroyEntry, createNode, destroyNode и другие, которые используются для управления памятью и основными операциями в дереве.

Класс реализует основные операции для работы с деревом, а также специальные методы для разделения и объединения узлов дерева.

Вложенный класс Entry представляет пару ключ-значение, key – указатель на ключ, а value – указатель на значение. В классе Node содержится информация о узле B+ дерева, включая ссылки на родительский, левый и правый узлы, массив элементов и массив дочерних узлов.

В классе BPlusTreeMap реализованы различные методы для работы с B+ деревом, включая создание и уничтожение узлов, разделение узлов при переполнении, поиск элементов, добавление новых элементов, изменение ключа минимального элемента листа и другие вспомогательные методы.

Класс предоставляет эффективные операции добавления, удаления и поиска элементов благодаря использованию B+ дерева. Эта структура данных особенно полезна при работе с большими объемами данных, где требуется быстрый доступ к элементам по ключу и эффективное использование памяти.

# Руководство пользователя

Пользователь может написать команды в файле, такие как ADD для добавления данных в базу, GET для проверки наличия данных в базе, CONTAINS для получения данных из базы, REMOVE для удаления данных из базы, REMOVE\_DATABASE для удаления всей базы, REMOVE\_SCHEMA для удаления схемы и REMOVE\_TABLE для удаления таблицы. База данных содержит таблицы с полями id, фамилия, имя, отчество, дата рождения, id hr менеджера, id контеста, язык программирования, число заданий, число решенных заданий и обнаружено списывание (true или false). При выполнении команд из файла, путь к файлу указывается пользователем через диалоговое окно. Пример представлен на рисунке 1.

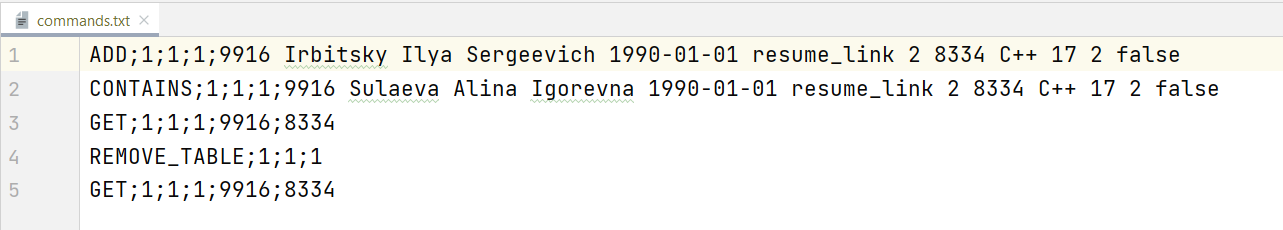
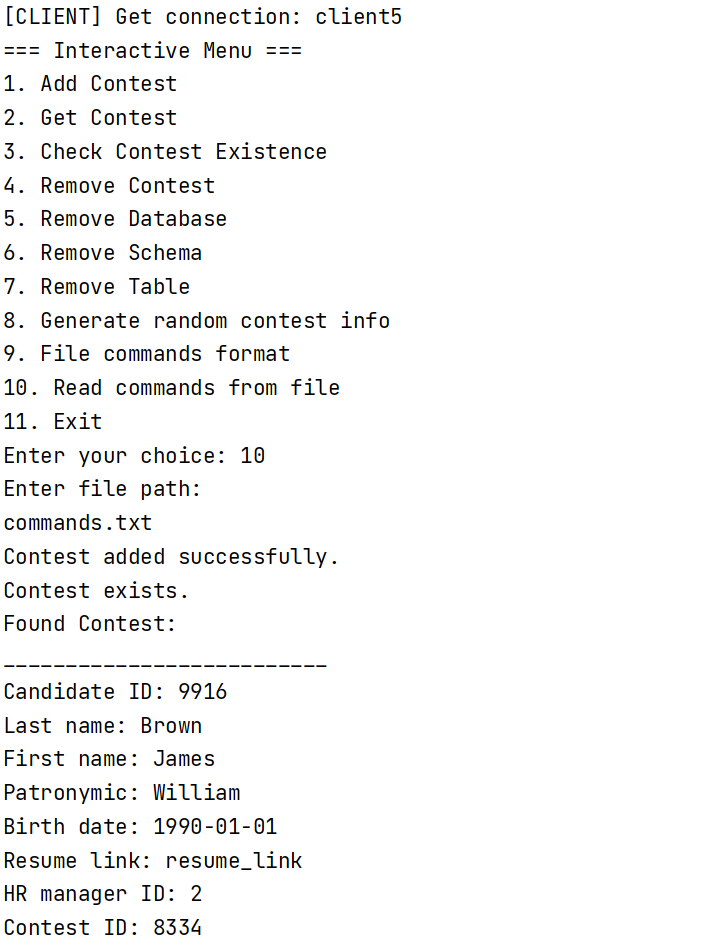


Рисунок 1. Пример команд в commands.txt

Пользователь имеет возможность наблюдать за результатом выполнения команд в консоли. Подробнее показано на рисунке 2



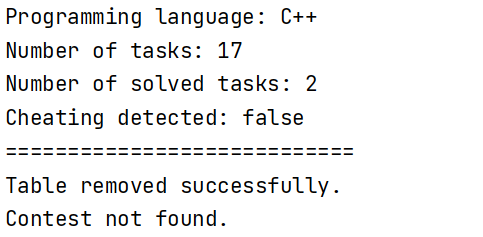


Рисунок 2. Вывод в консоли

Также пользователь может выполнять команды в консоли. Присутствует интерактивное меню. Для выбора нужной команды пользователь должен ввести соответствующую цифру, используя интерактивное меню. Подробнее показано в листинге 1.

|  |
| --- |
| Листинг 1. Демонстрация работы интерактивного меню |
| [CLIENT] Get connection: client2  === Interactive Menu ===  1. Add Contest  2. Get Contest  3. Check Contest Existence  4. Remove Contest  5. Remove Database  6. Remove Schema  7. Remove Table  8. Generate random contest info  9. File commands format  10. Read commands from file  11. Exit  Enter your choice: 8  8700 Williams James William 1990-01-01 resume\_link 3 1184 JavaScript 8 14 true  === Interactive Menu ===  1. Add Contest  2. Get Contest  3. Check Contest Existence  4. Remove Contest  5. Remove Database  6. Remove Schema  7. Remove Table  8. Generate random contest info  9. File commands format  10. Read commands from file  11. Exit  Enter your choice: 1  Enter database name: 1  Enter schema name: 1  Enter table name: 1  Enter contest info string: 8700 Williams James William 1990-01-01 resume\_link 3 1184 JavaScript 8 14 true  Contest added successfully.  === Interactive Menu ===  1. Add Contest  2. Get Contest  3. Check Contest Existence  4. Remove Contest  5. Remove Database  6. Remove Schema  7. Remove Table  8. Generate random contest info  9. File commands format  10. Read commands from file  11. Exit  Enter your choice: 2  Enter database name: 1  Enter schema name: 1  Enter table name: 1  Enter candidate ID: 8700  Enter contest ID: 1184  Found Contest:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Candidate ID: 8700  Last name: Williams  First name: James  Patronymic: William  Birth date: 1990-01-01  Resume link: resume\_link  HR manager ID: 3  Contest ID: 1184  Programming language: JavaScript  Number of tasks: 8  Number of solved tasks: 14  Cheating detected: true  ============================  === Interactive Menu ===  1. Add Contest  2. Get Contest  3. Check Contest Existence  4. Remove Contest  5. Remove Database  6. Remove Schema  7. Remove Table  8. Generate random contest info  9. File commands format  10. Read commands from file  11. Exit  Enter your choice: 8  3861 Smith Michael Joseph 1990-01-01 resume\_link 2 8681 C++ 7 19 false  === Interactive Menu ===  1. Add Contest  2. Get Contest  3. Check Contest Existence  4. Remove Contest  5. Remove Database  6. Remove Schema  7. Remove Table  8. Generate random contest info  9. File commands format  10. Read commands from file  11. Exit  Enter your choice: 1  Enter database name: 1  Enter schema name: 2  Enter table name: 2  Enter contest info string: 3861 Smith Michael Joseph 1990-01-01 resume\_link 2 8681 C++ 7 19 false  Contest added successfully.  === Interactive Menu ===  1. Add Contest  2. Get Contest  3. Check Contest Existence  4. Remove Contest  5. Remove Database  6. Remove Schema  7. Remove Table  8. Generate random contest info  9. File commands format  10. Read commands from file  11. Exit  Enter your choice: 3  Enter database name: 1  Enter schema name: 2  Enter table name: 2  Enter contest info string: 3861 Smith Michael Joseph 1990-01-01 resume\_link 2 8681 C++ 7 19 false  Contest exists.  === Interactive Menu ===  1. Add Contest  2. Get Contest  3. Check Contest Existence  4. Remove Contest  5. Remove Database  6. Remove Schema  7. Remove Table  8. Generate random contest info  9. File commands format  10. Read commands from file  11. Exit  Enter your choice: 6  Enter database name: 1  Enter schema name: 2  Schema removed successfully.  === Interactive Menu ===  1. Add Contest  2. Get Contest  3. Check Contest Existence  4. Remove Contest  5. Remove Database  6. Remove Schema  7. Remove Table  8. Generate random contest info  9. File commands format  10. Read commands from file  11. Exit  Enter your choice: 3  Enter database name: 1  Enter schema name: 2  Enter table name: 2  Enter contest info string: 3861 Smith Michael Joseph 1990-01-01 resume\_link 2 8681 C++ 7 19 false  Contest does not exist.  === Interactive Menu ===  1. Add Contest  2. Get Contest  3. Check Contest Existence  4. Remove Contest  5. Remove Database  6. Remove Schema  7. Remove Table  8. Generate random contest info  9. File commands format  10. Read commands from file  11. Exit  Enter your choice: 3  Enter database name: 1  Enter schema name: 2  Enter table name: 2  Enter contest info string: 8700 Williams James William 1990-01-01 resume\_link 3 1184 JavaScript 8 14 true  Contest does not exist.  === Interactive Menu ===  1. Add Contest  2. Get Contest  3. Check Contest Existence  4. Remove Contest  5. Remove Database  6. Remove Schema  7. Remove Table  8. Generate random contest info  9. File commands format  10. Read commands from file  11. Exit  Enter your choice: 3  Enter database name: 1  Enter schema name: 1  Enter table name: 1  Enter contest info string: 8700 Williams James William 1990-01-01 resume\_link 3 1184 JavaScript 8 14 true  Contest exists.  === Interactive Menu ===  1. Add Contest  2. Get Contest  3. Check Contest Existence  4. Remove Contest  5. Remove Database  6. Remove Schema  7. Remove Table  8. Generate random contest info  9. File commands format  10. Read commands from file  11. Exit  Enter your choice: 4  Enter database name: 1  Enter schema name: 1  Enter table name: 1  Enter contest info string: 8700 Williams James William 1990-01-01 resume\_link 3 1184 JavaScript 8 14 true  Contest removed successfully.  === Interactive Menu ===  1. Add Contest  2. Get Contest  3. Check Contest Existence  4. Remove Contest  5. Remove Database  6. Remove Schema  7. Remove Table  8. Generate random contest info  9. File commands format  10. Read commands from file  11. Exit  Enter your choice: 11  Exiting...  Process finished with exit code 0 |



# Вывод

В ходе выполнения курсового проекта была создана программа, которая позволяет клиенту и серверу обмениваться данными через разделяемую память. Для этого были разработаны классы для обработки запросов и команд, хранения и передачи данных, а также установления соединения между клиентом и сервером.

Программа эффективно использует память, включая пул строк для оптимизации ресурсов. Взаимодействие между клиентом и сервером осуществляется через классы, которые позволяют отправлять и получать запросы и ответы. Для установления соединения используется класс, который обеспечивает передачу сообщений через разделяемую память.

Также был реализован сервер для обработки запросов, который может выполнить запросы самостоятельно или отправить их на соответствующий сервер базы данных. Сервер также отправляет ответы клиенту и обеспечивает синхронизацию при необходимости.

Кроме того, в программе предусмотрен механизм регистрации событий, который принимает сообщения через разделяемую память.

В целом, разработанная программа обеспечивает эффективное взаимодействие между клиентом и сервером, обеспечивая передачу данных через разделяемую память и обработку запросов к базе данных. Она является гибкой и позволяет легко добавлять новую функциональность при необходимости.

Список использованных источников

1. Александреску А. Современное проектирование на C++. - М.: Вильямс, 2019. - 336 с.
2. Мейерс C. Эффективный и современный C++. - М.: Вильямс, 2016. - 306 с.
3. Шилдт Г. Самоучитель C++. - СПб.: БХВ-Петербург, 2005. - 688 с.

# Приложение

<https://github.com/lesopylka/courseprojectonfundamentalinfomartics>