Университет ИТМО

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.01 Информатика и вычислительная техника

Лабораторная работа №3

по дисциплине «Низкоуровневое программирование»
Вариант Protobuf

Выполнил:
Студент группы P33302
Овчаренко А.А.

Преподаватель:
Кореньков Юрий Дмитриевич

г. Санкт-Петербург
2023

Содержание

Задание *3*

Библиотека для работы с protobuf *5*

Пример сеанса работы разработанных программ *6*

Описание решения *9*

Протокол *11*

Демонстрация работы операции выборки *13*

- Данные 13
- Запрос на сервер 13
- Ответ с сервера 13
- Запрос в базу данных neo4j и ответ 18

Итог *19*

Задание

На базе данного транспортного формата описать схему протокола обмена информацией и воспользоваться существующей библиотекой по выбору для реализации модуля, обеспечивающего его функционирование. Протокол должен включать представление информации о командах создания, выборки, модификации и удаления данных в соответствии с данной формой, и результатах их выполнения. Используя созданные в результате выполнения заданий модули, разработать в виде консольного приложения две программы: клиентскую и серверную части. Серверная часть – получающая по сети запросы и операции описанного формата и последовательно выполняющая их над файлом данных с помощью модуля из первого задания. Имя фала данных для работы получать с аргументами командной строки, создавать новый в случае его отсутствия. Клиентская часть – в цикле получающая на стандартный ввод текст команд, извлекающая из него информацию о запрашиваемой операции с помощью модуля из второго задания и пересылающая её на сервер с помощью модуля для обмена информацией, получающая ответ и выводящая его в человеко-понятном виде в стандартный вывод.
Порядок выполнения:

1. Изучить выбранную библиотеку
    a. Библиотека должна обеспечивать сериализацию и десериализацию с валидацией в соответствии со схемой
    b. Предпочтителен выбор библиотек, поддерживающих кодогенерацию на основе схемы
    c. Библиотека может поддерживать передачу данных посредством TCP соединения
        i. Иначе, использовать сетевые сокеты посредством API ОС
    d. Библиотека может обеспечивать диспетчеризацию удалённых вызовов
        i. Иначе, реализовать диспетчеризацию вызовов на основе информации о виде команды
2. На основе существующей библиотеки реализовать модуль, обеспечивающий взаимодействие
    a. Описать схему протокола в поддерживаемом библиотекой формате
        i. Описание должно включать информацию о командах, их аргументах и результатах
        ii. Схема может включать дополнительные сущности (например, для итератора)
    b. Подключить библиотеку к проекту и сформировать публичный интерфейс модуля с использованием встроенных или сгенерированных структур данных используемой библиотеки

        i. Поддержать установление соединения, отправку команд и получение их результатов
        ii. Поддержать приём входящих соединений, приём команд и отправку их результатов
    c. Реализовать публичный интерфейс посредством библиотеки в соответствии с п1
3. Реализовать серверную часть в виде консольного приложения
    a. В качестве аргументов командной строки приложение принимает:
        i. Адрес локальной конечной точки для прослушивания входящих соединений
        ii. Имя файла данных, который необходимо открыть, если он существует, иначе создать
    b. Работает с файлом данных посредством модуля из задания 1
    c. Принимает входящие соединения и взаимодействует с клиентами посредством модуля из п2
    d. Поступающая информация о запрашиваемых операциях преобразуется из структур данных модуля взаимодействия к структурам данных модуля управления данными и наоборот
4. Реализовать клиентскую часть в виде консольного приложения
    a. В качестве аргументов командной строки приложение принимает адрес конечной точки для подключения
    b. Подключается к серверу и взаимодействует с ним посредством модуля из п2
    c. Читает со стандартного ввода текст команд и анализирует их посредством модуля из задания 2
    d. Преобразует результат разбора команды к структурам данных модуля из п2, передаёт их для обработки на сервер, возвращаемые результаты выводит в стандартный поток вывода
5. Результаты тестирования представить в виде отчёта, в который включить:
    a. В части 3 привести пример сеанса работы разработанных программ
    b. В части 4 описать решение, реализованное в соответствии с пп.2-4
    c. В часть 5 включить составленную схему п.2а

Библиотека для работы с protobuf

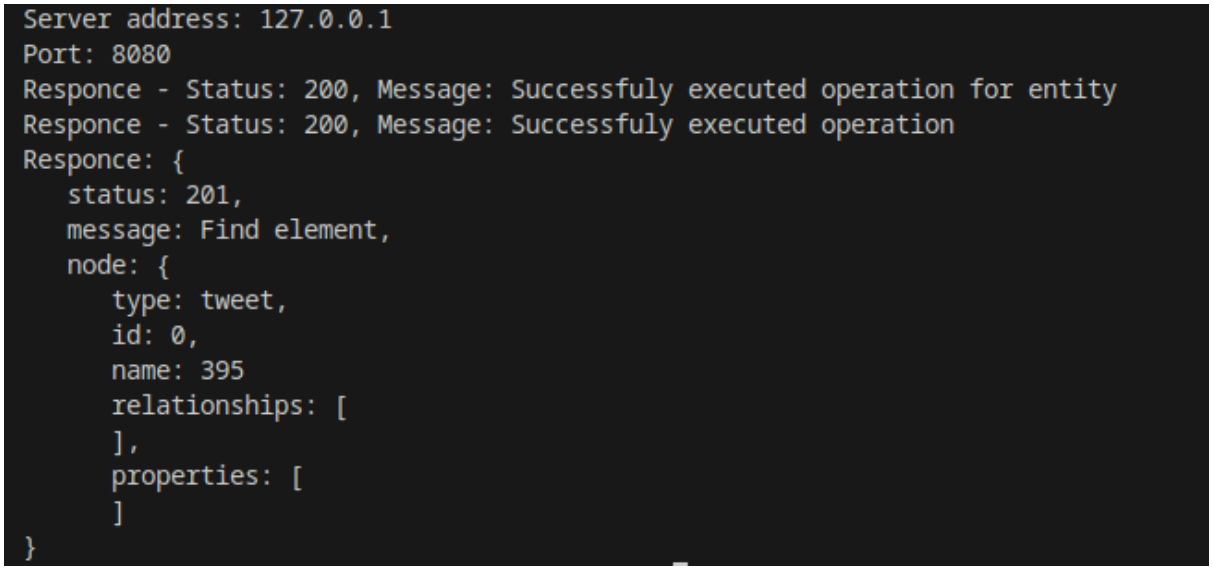
Для обмена данными между клиентом и сервером по варианту нужно было использовать протокол protobuf. Для работы с ним на языке с была выбрана библиотека protobuf-c. Библиотека удовлетворяет всем требованиям для работы с протоколом protobuf. Она предоставляет интерфейс для генерации с-структур, методов сериализации и десериализации.

Пример сеанса работы разработанных программ

Клиентский модуль при запуске требует указания адреса сервера и порта для подключения. Также дополнительно был реализован выбор источника данных: командная строка или файл. В примере использовался файл в качестве источника данных. Финальная команда для запуска - ./main 127.0.0.1 8080 0 simple
Файл содержит следующие запросы

```
{
   type_of_element:node,
   value_type:void,
   name:tweet
}
mutation CreateEntity(i: $i) {
   createEntity(i: $i) {
      id
   }
}
;
{
   type:tweet,
   name:395
}
mutation CreateNode(i: $i) {
   createNode(i: $i) {
      id
   }
}
;
{}
query SelectNode(i: $i) {
   selectNode(i: $i) {
      type(type: tweet)
   }
}
;
```

Серверный модуль требует указания порта, на который принимает запросы на подключения, и путь к файлу. Финальная команда для запуска -./llp 8080 database
Результат выполнения запросов:



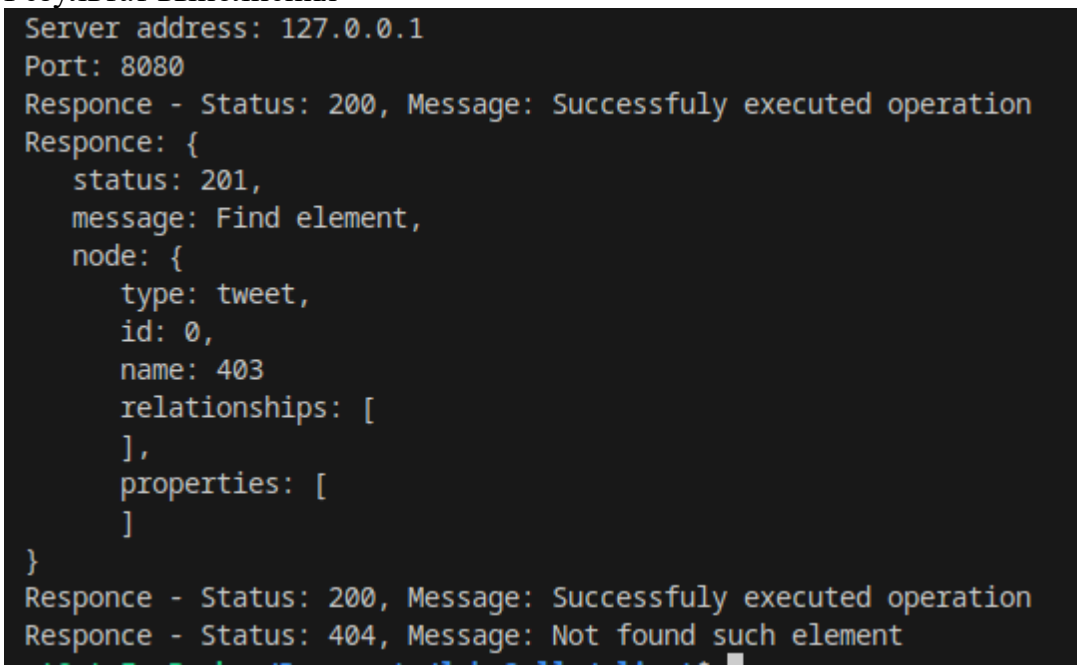
Пояснения: первый запрос – запрос на создание сущности tweet типа Node, второй запрос на создание объекта сущности tweet, третий запрос на получение всех объектов сущности tweet.

Пример работы операций UPDATE и DELETE. Содержание файла:

```
{
   type:tweet,
   name:403
}
mutation UpdateNode(i: $i) {
   updateNode(i: $i) {
      id(id: 0)
      type(type: tweet)
   }
}
;
{}
query SelectNode(i: $i) {
   selectNode(i: $i) {
      type(type: tweet)
   }
}
;
{}
mutation DeleteNode(i: $i) {
   deleteNode(i: $i) {
      id(id: 0)
      type(type: tweet)
   }
}
;
{}
query SelectNode(i: $i) {
```

```
    selectNode(i: $i) {
        type(type: tweet)
    }
}
;
```

Результат выполнения



Описание решения

Как было отмечено выше, библиотека protobuf-c предоставляет средства для генерации структур и методов. Шаблон передаваемых данных описывается в proto-файлах, используется синтаксис proto2. Пример шаблона сообщения – запрос на сервер:

```
syntax = "proto2";

import "operation.proto";
import "filter.proto";
import "entity.proto";
import "node.proto";
import "relationship.proto";
import "property.proto";
import "typeelement.proto";
import "iterator.proto";

message RequestMessage {
    required OperationProto operation = 1;
    required TypeOfElementProto type = 2;
    optional FilterMessage filter = 3;

    optional EntityMessage entity = 4;
    optional NodeMessage node = 5;
    optional RelationshipMessage relationship = 6;
    optional PropertyMessage property = 7;

    repeated string relationships = 8;
    repeated string properties = 9;

    optional IteratorMessage iterator = 10;
}
```

Клиентское приложение, используя модуль из лабораторной работы 2, производит анализ ввода пользователя, который происходит на языке GraphQl, создает AST дерево. После этого происходит конвертация из AST дерева в структуры данных, описанных через proto-файлы, по средствам обхода дерева в глубину и составляется запрос. Далее используются сгенерированные protobuf-c методы для представления запроса в бинарном формате.

```
request = parse_ast(&root);
len = request_message__get_packed_size(request);
buf = malloc(len);

request_message__pack(request, buf);
send_net(len, buf, client_fd);
```

Все необходимые структуры для общения между клиентом и сервером находятся в директории proto, ознакомиться можно перейдя на github-репозиторий.

Для общения между клиентом и сервером был реализован модуль net. В нем содержатся необходимы команды для работы с сокетами: открытие сокета, установление соединения, прием сообщения, отправка сообщения, закрытие сокета. Протокол protobuf передает данные в бинарном формате. После приема сообщения, используются сгенерированные библиотекой методы для конвертации бинарных данных в с-структуры.

```
buffer = (uint8_t *) malloc(sizeof(uint8_t) * 4096);
length = receive_from_socket(client_fd, buffer);
responce = responce_message__unpack(NULL, length, buffer);
```

На сервере происходит конвертация полученных данных в элементы модуля базы данных, разработанного в первой лабораторной работе, определения типа операции и использование интерфейса для выполнения требуемой операции.

Операция выборки работает следующим образом:

- Поиск требуемого объекта в соответствии с условиями из структуры фильтр
- Поиск запрашиваемых связей с другими объектами
- Поиск требуемых свойств объекта
- Сохранения состояния итератора: read_block и offset. Это нужно, чтобы клиент мог запросить следующий элемент.
- Отправка клиенту объекта со связными объектами и свойствами, а также состояния итератора.

Протокол

Протокол обмена данными определяется структурами, описанными через proto-файлы, так как был использован protobuf в качестве протокола сериализации данных.
Опишем две основные структуры RequestMessage и ResponceMessage.

Описание RequestMessage

- OperationProto operation – операция - создание/обновление/удаление/выбор объекта сущности, создание/удаление сущности
- TypeOfElementProto type – тип элемента, используется для работы с объектами сущностей: Node, Relationship, Property
- FilterMessage filter – фильтр для выбора элементов. Нужен в операциях обновления, удаления и выборки
- EntityMessage entity – сущность, которую хотим создать или удалить
- NodeMessage node – объект типа Node, которых хотим создать, или на которых хотим обновить уже существующий
- RelationshipMessage relationship - объект типа Relationship, которых хотим создать, или на которых хотим обновить уже существующий
- PropertyMessage property - объект типа Property, которых хотим создать, или на которых хотим обновить уже существующий
- string relationships - список запрашиваемых связей. Поле нужно для выполнения операции выборки, используется, чтобы взять объекты типа Node, с которыми исходный объект находится в указанной связи
- string properties - список запрашиваемых свойств объекта
- IteratorMessage iterator – итератор используется для итерирования между объектами операции выборки

Описание ResponceMessage

- int32 status – статус операции: 200 – успешное выполнение, 201 – ответ содержит объекты для вывода пользователю, 400 – некорректный запрос, 404 – запрашиваемый объект не найден

- string message – сообщение
- NodeResponce node – запрашиваемый объект
- IteratorMessage iterator – итератор для выполнения запроса на получение следующего объекта

Более детально со структурами обмена данными, а также с реализацией клиентского и серверного модулей можно ознакомиться, перейдя по ссылке на github-репозиторий.

Демонстрация работы операции выборки

- Данные

В качестве данных для базы данных использовался готовый датасет twitter-v2 из примера для базы данных neo4j. Ссылка на github-репозиторий датасета. Датасет доступен по ссылке (имя пользователя “twitter”, пароль “twitter”, название датасета “twitter”).

- Запрос на сервер

```
{}
query SelectNode(i: $i) {
  selectNode(i: $i) {
        id(id: 0)
    type(type: tweet)
    relationships {
      reply_to
      mentioned
    }
    properties {
      text
      created_at
    }
  }
}
;
```

- Ответ с сервера

```
Server address: 127.0.0.1
Port: 8080
Responce: {
  status: 201,
  message: Find element,
  node: {
    type: tweet,
    id: 0,
    name: 395
    relationships: [
      {
        relation_type: reply_to,
```

```
      type: tweet,
      id: 12,
      name: 342
      {
        value_type: 2,
        type: text,
        value: @anshublog @martin_casado Oooh, another big one: Graph Database, thanks @neo4j &amp;
@emileifrem !
      }
    }
    ,
    {
      relation_type: reply_to,
      type: tweet,
      id: 13,
      name: 341
      {
        value_type: 2,
        type: text,
        value: @anshublog @martin_casado Oooh, another big one: Graph Database, thanks @neo4j &amp;
@emileifrem !
      }
    }
    ,
    {
      relation_type: reply_to,
      type: tweet,
      id: 14,
      name: 0
      {
        value_type: 2,
        type: text,
        value: @anshublog @martin_casado Oooh, another big one: Graph Database, thanks @neo4j &amp;
@emileifrem !
      }
    }
    ,
    {
      relation_type: reply_to,
      type: tweet,
```

```
      id: 1,
      name: 397
      {
        value_type: 2,
        type: text,
        value: @martin_casado @anshublog @neo4j @emileifrem True, 15~ years ago.. I think you could just focus on the last decade… https://t.co/lkyGb5kzaE
      }
    }
    ,
    {
      relation_type: reply_to,
      type: tweet,
      id: 15,
      name: 339
      {
        value_type: 2,
        type: text,
        value: @anshublog @martin_casado Oooh, another big one: Graph Database, thanks @neo4j &amp;amp; @emileifrem !
      }
    }
    ,
    {
      relation_type: reply_to,
      type: tweet,
      id: 12,
      name: 342
      {
        value_type: 2,
        type: text,
        value: @anshublog @martin_casado Oooh, another big one: Graph Database, thanks @neo4j &amp;amp; @emileifrem !
      }
    }
    ,
    {
      relation_type: reply_to,
      type: tweet,
      id: 13,
```

```
      name: 341
      {
        value_type: 2,
        type: text,
        value: @anshublog @martin_casado Oooh, another big one: Graph Database, thanks @neo4j &amp;
@emileifrem !
      }
    }
    ,
    {
      relation_type: reply_to,
      type: tweet,
      id: 14,
      name: 0
      {
        value_type: 2,
        type: text,
        value: @anshublog @martin_casado Oooh, another big one: Graph Database, thanks @neo4j &amp;
@emileifrem !
      }
    }
    ,
    {
      relation_type: reply_to,
      type: tweet,
      id: 1,
      name: 397
      {
        value_type: 2,
        type: text,
        value: @martin_casado @anshublog @neo4j @emileifrem True, 15~ years ago.. I think you could just
focus on the last decade… https://t.co/lkyGb5kzaE
      }
    }
    ,
    {
      relation_type: reply_to,
      type: tweet,
      id: 15,
      name: 339
```

```
          {
            value_type: 2,
            type: text,
            value: @anshublog @martin_casado Oooh, another big one: Graph Database, thanks @neo4j &amp;amp;
@emileifrem !
          }
        }
      ],
      properties: [
        {
          value_type: 2,
          type: text,
          value: @asynchio @anshublog @neo4j @emileifrem Yup. I have my own list. Just curious what others
think ...
        }
        ,
        {
          value_type: 2,
          type: text,
          value: @asynchio @anshublog @neo4j @emileifrem Yup. I have my own list. Just curious what others
think ...
        }
        ,
        {
          value_type: 2,
          type: created_at,
          value: 2021-03-13T17:23:42Z
        }
        ,
        {
          value_type: 2,
          type: created_at,
          value: 2021-03-13T17:23:42Z
        }
      ]
    }
}
```

- Запрос в базу данных neo4j и ответ

```
MATCH (n:Tweet)-[:REPLY_TO]->(r) WHERE n.id IN [ 1370787667927912448]
RETURN ID(n), n.text, n.created_at, ID(r)
```

```
╒═══════╤═══════════════════════════════════════════════════╤═══════════════════════╤═══════╕
│ID(n)  │n.text                                             │n.created_at           │ID(r)  │
╞═══════╪═══════════════════════════════════════════════════╪═══════════════════════╪═══════╡
│395    │"@asynchio @anshublog @neo4j @emileif rem Yup. I have my own list. Just cur│"2021-03-13T17:23:42Z"│342    │
│       │ious what others think ..."                        │                       │       │
├───────┼───────────────────────────────────────────────────┼───────────────────────┼───────┤
│395    │"@asynchio @anshublog @neo4j @emileif rem Yup. I have my own list. Just cur│"2021-03-13T17:23:42Z"│341    │
│       │ious what others think ..."                        │                       │       │
├───────┼───────────────────────────────────────────────────┼───────────────────────┼───────┤
│395    │"@asynchio @anshublog @neo4j @emileif rem Yup. I have my own list. Just cur│"2021-03-13T17:23:42Z"│0      │
│       │ious what others think ..."                        │                       │       │
├───────┼───────────────────────────────────────────────────┼───────────────────────┼───────┤
│395    │"@asynchio @anshublog @neo4j @emileif rem Yup. I have my own list. Just cur│"2021-03-13T17:23:42Z"│397    │
│       │ious what others think ..."                        │                       │       │
├───────┼───────────────────────────────────────────────────┼───────────────────────┼───────┤
│395    │"@asynchio @anshublog @neo4j @emileif rem Yup. I have my own list. Just cur│"2021-03-13T17:23:42Z"│339    │
│       │ious what others think ..."                        │                       │       │
└───────┴───────────────────────────────────────────────────┴───────────────────────┴───────┘
```

Итог

В рамках выполнения лабораторной работы был изучен протокол сериализации структурированных данных Protocol Buffers, была изучена библиотека protobuf-c для работы с данным протоколом на языке программирования C. Были разработаны клиентский и серверный модель для анализа запроса пользователя и обработки его в базе данных.