Университет ИТМО

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Информатика и вычислительная техника
Дисциплина «Низкоуровневое программирование»

Отчет
По лабораторной работе №3
Вариант 3 (Protocol Buffers)

Выполнил:
Чернова А. И.
Преподаватель:
Кореньков Ю. Д.

Санкт-Петербург, 2024 г.

Цель: 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.

Задачи:

1. Выбрать библиотеку для реализации protocol buffers.
2. Разработать в виде консольного приложения две программы: клиентскую и серверную части.
3. В серверной части получать по сети запросы и операции описанного формата и последовательно выполнять их над файлом данных с помощью модуля из первого задания. Имя файла данных для работы получать с аргументами командной строки, создавать новый в случае его отсутствия.
4. В клиентской части в цикле получать на стандартный ввод текст команд, извлекать из него информацию о запрашиваемой операции с помощью модуля из второго задания и пересылать её на сервер с помощью модуля для обмена информацией, получать ответ и выводить его в человеко-понятном виде в стандартный вывод.

Описание работы:

Программа представляет собой совокупность артефактов, полученных по результатам первой (модуль db) и второй (модуль parser) лабораторной работы. Для сериализации передаваемых данных был добавлен модуль proto (от protocol buffers). Сборка проекта осуществляется с помощью Cmake, а внешне – с помощью скрипта build.sh, который дополнительно пересобирает файлы Yacc и Lex (не разобрался как включить их сборку в Cmake). По итогу мы получаем два исполняемых файла: LLP3_server и LLP3_client. Способ запуска сервера идентичен способу в первой лабораторной (флаги -p (parsing), -n (new), -o (open)), а для запуска клиента нужно лишь указать адрес подключения в единственном аргументе командной строки.

Аспекты реализации:

Так как изначально библиотека protobuf не ориентирована на язык C (только на C++), было решено использовать ее адаптацию – библиотеку nanopb. Для работы она требует оформления .proto – файла, в котором должно быть описана структура передаваемых «объектов». В моем случае эта структура (файл message.proto) полностью повторяет структуру дерева запроса из второй лабораторной:

```
message Query_tree {
    required int32 command = 1; repeated Filter filters
    = 2; repeated Value_setting settings = 3;

    message Filter {
                repeated Comparator comp_list = 1;
    }

    message Value_setting {
                 required Field_value_pair fv = 1;
    }

    message Comparator {
        required int32 operation = 1; required
        Field_value_pair fv = 2;
    }

    message Field_value_pair { required string
        field = 1; required int32 val_type = 2;
        required int64 int_val = 3; required float
        real_val = 4;required string str_val = 5;
    }
}
```

Помимо дерева, файл содержит описание структуры «ответа» сервера:

```
message   Response{ required int32
    last = 2;
    required string r_string = 1;
}
```

Сервер возвращает результат выполнения запроса в виде строки – это может быть описание ошибки, либо тело успешного ответа, при передаче это не имеет значения. Так как размер ответа может быть огромным (например, выборка всех элементов), было принято решение ограничить одну порцию ответа до 1024 символов, и отправлять результат именно такими частями, постоянно обрезая строку слева. Для этого введено поле last, которое обозначает финальный блок ответа.

Передача по сети была организована посредством сетевых сокетов API ОС (пример для стороны клиента) :

```
struct sockaddr_in servaddr;char *path =
NULL;

<…>

sockfd = socket(AF_INET, SOCK_STREAM, 0);

memset(&servaddr, 0, sizeof(servaddr)); servaddr.sin_family =
AF_INET; servaddr.sin_addr.s_addr = inet_addr(argv[1]);
servaddr.sin_port = htons(PORT);

if (connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) != 0){perror("connect");
        return 1;
}
```

Кодирование и декодирование данных в protocol buffers выглядит таким образом (пример для стороны сервера):

```
 Query_tree t = {};
 if (!pb_decode_delimited(&input, Query_tree_fields, &t))
 {
        printf("Decode failed: %s\n", PB_GET_ERROR(&input));return 2;
 }

 handle_query(file, t, &response);Response r =

 {};

 <…>

 if (!pb_encode_delimited(&output, Response_fields, &r))
 {
        fprintf(stderr, "Encoding failed: %s\n", PB_GET_ERROR(&output));
 }
```

Результаты:

- Инициализация файла на сервере (и парсинг файла с генератором) и подключение пользователя

```
user@LAPTOP-M8TDLIKV:/mnt/d/LLP_lab3$ ./LLP3_server -p file.txt db/out.txt
Initializing pattern.
Input the number of fields in pattern: 4
--- Field 0   ---
Enter field name: age
0. Boolean
1. Integer
2. Float
3. String
Choose field type: 1
--- Field 1   ---
Enter field name: name
0. Boolean
1. Integer
2. Float
3. String
Choose field type: 3
--- Field 2   ---
Enter field name: male
0. Boolean
1. Integer
2. Float
3. String
Choose field type: 0
--- Field 3   ---
Enter field name: balance
0. Boolean
1. Integer
2. Float
3. String
Choose field type: 2
server: waiting for connection…
```

- Добавление нового элемента и его выборка

```
user@LAPTOP-M8TDLIKV:/mnt/d/LLP_lab3$ ./LLP3_client 127.0.0.1
client: connecting
db.insert({parent:1}, {name:"slavik", age:123, male:1, balance:99.13})
Added successfully.
db.find({name:"slavik"})
--- FIND RESULT ---
--- TUPLE 100 ---
age                     123
name                    slavik
male                    1
balance                 99.129997
```

- Демонстрация работы комбинаций булевских выражений

```
db.find({age: {$ne:123}, $or[age: 123, name: "ugand"]})
--- FIND RESULT ---
--- TUPLE  95 ---
age                     3126
name                    ugand
male                    1
balance                 2.422918
```

```
db.find({$or[name:"slavik", age:{$lt:800}]})
--- FIND RESULT ---
--- TUPLE 100 ---
age                     123
name                    slavik
male                    1
balance                 99.129997
--- TUPLE  60 ---
age                     734
name                    yybvhafp
male                    1
balance                 25.390679
```

- Обновление элемента (+ поиск по id)

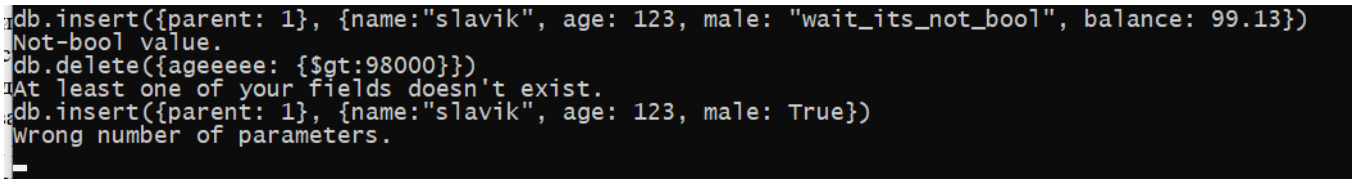
```
db.update({name:"slavik"}, $set:{name: "stanislavik"})
--- UPDATE RESULT ---
Updated id: 100
db.find({id:100})
--- FIND RESULT ---
--- TUPLE 100 ---
age                     123
name                    stanislavik
male                    1
balance                 99.129997
```

```
db.update({age: {$gt:98000}}, $set:{name: "old_male", male: True})
--- UPDATE RESULT ---
Updated id: 83
Updated id: 80
Updated id: 34
Updated id: 26
db.find({age: {$gt:98000}})
--- FIND RESULT ---
--- TUPLE  83 ---
age                     99036
name                    old_male
male                    1
balance                 41.154300
--- TUPLE  80 ---
age                     99482
name                    old_male
male                    1
balance                 21.434780
--- TUPLE  34 ---
age                     99893
name                    old_male
male                    1
balance                 8.046914
--- TUPLE  26 ---
age                     98358
name                    old_male
male                    1
balance                 49.953784
```

- Удаление элементов

```
db.delete({age: {$gt:98000}})
--- REMOVE RESULT ---
Removed successfully id: 83
Removed successfully id: 80
Removed successfully id: 34
Removed successfully id: 26
db.find({age: {$gt:98000}})
--- NO RESULTS ---
```

- Примеры обработанных ошибок



Выводы:

По итогам выполнения работы я научилась интегрировать малоизвестные библиотеки (с недостаточной документацией) в проект, находить ответы на интересующие меня вопросы в их исходном коде. Кроме того, на базовом уровне (на уровне локальной машины) была реализована сетевая передача данных на низком уровне, что, кажется, не является прямой задачей языка Си.