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:
Кузнецов Н. Д.
Преподаватель:
Кореньков Ю. Д.

Санкт-Петербург, 2023 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 server_module) и второй (модуль gremlin_module) лабораторной работы. Для сериализации передаваемых данных был добавлен модуль proto_module. Сборка проекта осуществляется с помощью Cmake. По итогу мы получаем два исполняемых файла: LLP3_server и LLP3_client. Серверный исполняемый файл принимает в качестве аргумента командной строки имя файла данных, а для запуска клиента нужно указать адрес и порт подключения. Если файл данных не существует, буден создан новый и инициализирован стандартным паттерном.

Аспекты реализации:

Для реализации Protocol Buffers на языке Си было принято решение использовать библиотеку nanopb, так как она имеет хорошую документацию с примерами использования (в том числе для передачи данных по сети). 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 Field_value{
  optional string str_val = 1;
  optional int64 int_val = 2;
  optional int64 bool_val = 3;
  optional double real_val = 4;
}

message Field{
  required uint32 type = 1;
  required string name = 2;
  required Field_value val = 3;
}

message Entity{
  repeated Field fields = 1;
  repeated uint64 rel = 2;
}

message Condition{
  required uint32 is_id = 1;
  required uint64 id = 2;
  required uint32 type = 3;
  required uint32 op = 4;
  required string field_name = 5;
  required Field_value val = 6;
}

message Filter{
  required uint32 negative = 1;
  repeated Condition conditions = 2;
}

message List_level{
  repeated Filter filters = 1;
}

message View {
  required uint32 op = 1;
  repeated List_level tree = 2;
  optional Entity entity = 3;
}
```

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

```
message Response{
  required uint32 is_last = 1;
  required string answer = 2;
}
```

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 64 символов (ограничение установлено в файле message.options; значение могло быть и больше, но так как конфигурационный файл применяется ко *всем* полям-массивам всех структур, в том числе дерева запроса, то оно заняло бы очень много места и не отправилось вообще), и отправлять результат именно такими частями, постоянно обрезая строку слева. Для этого введено поле `is_last`, которое обозначает финальный блок ответа.

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

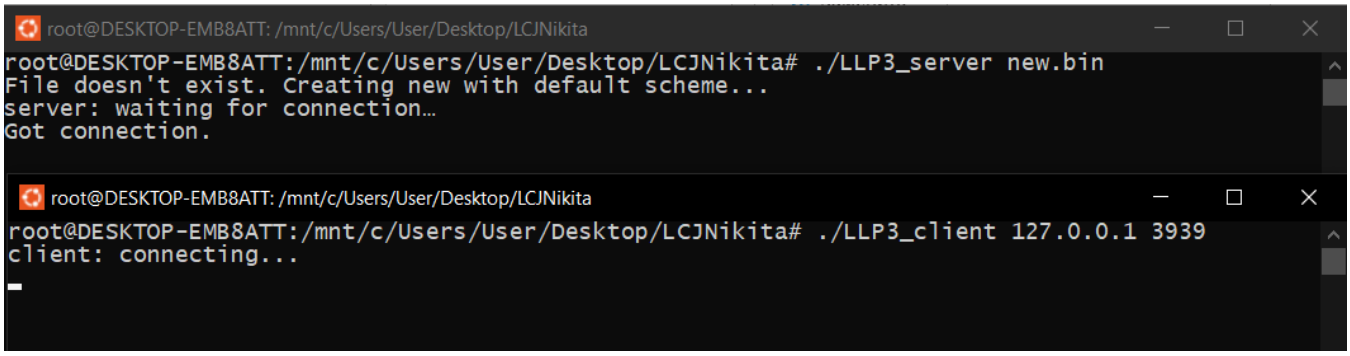
```
struct sockaddr_in servaddr;

<…>

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):

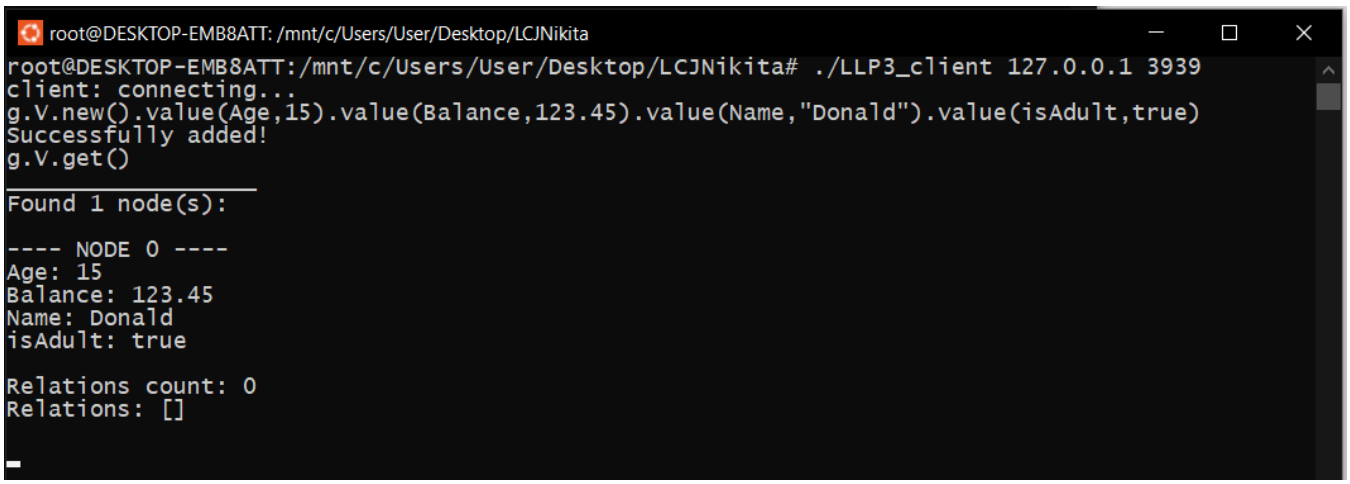
```
response = "";
View v = {};
if (!pb_decode_delimited(&input, View_fields, &v)) {
    printf("Decode failed: %s\n", PB_GET_ERROR(&input));
    return 2;
}

<…handling request…>

Response r = {};

<…>

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

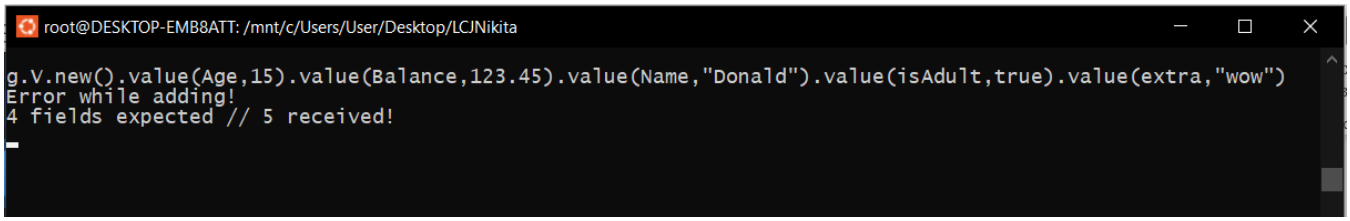
Результаты:

- Инициализация файла на сервере и подключение пользователя

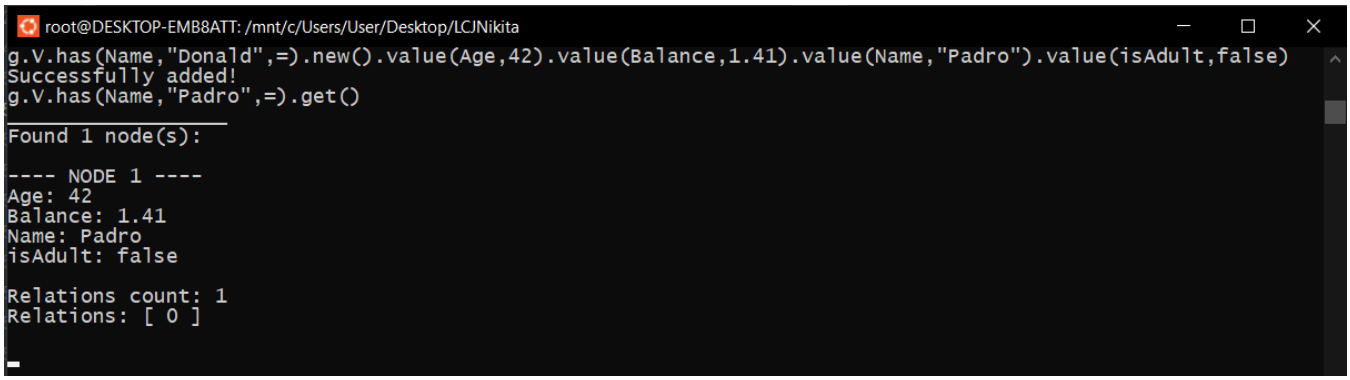


- Добавление нового элемента и его выборка

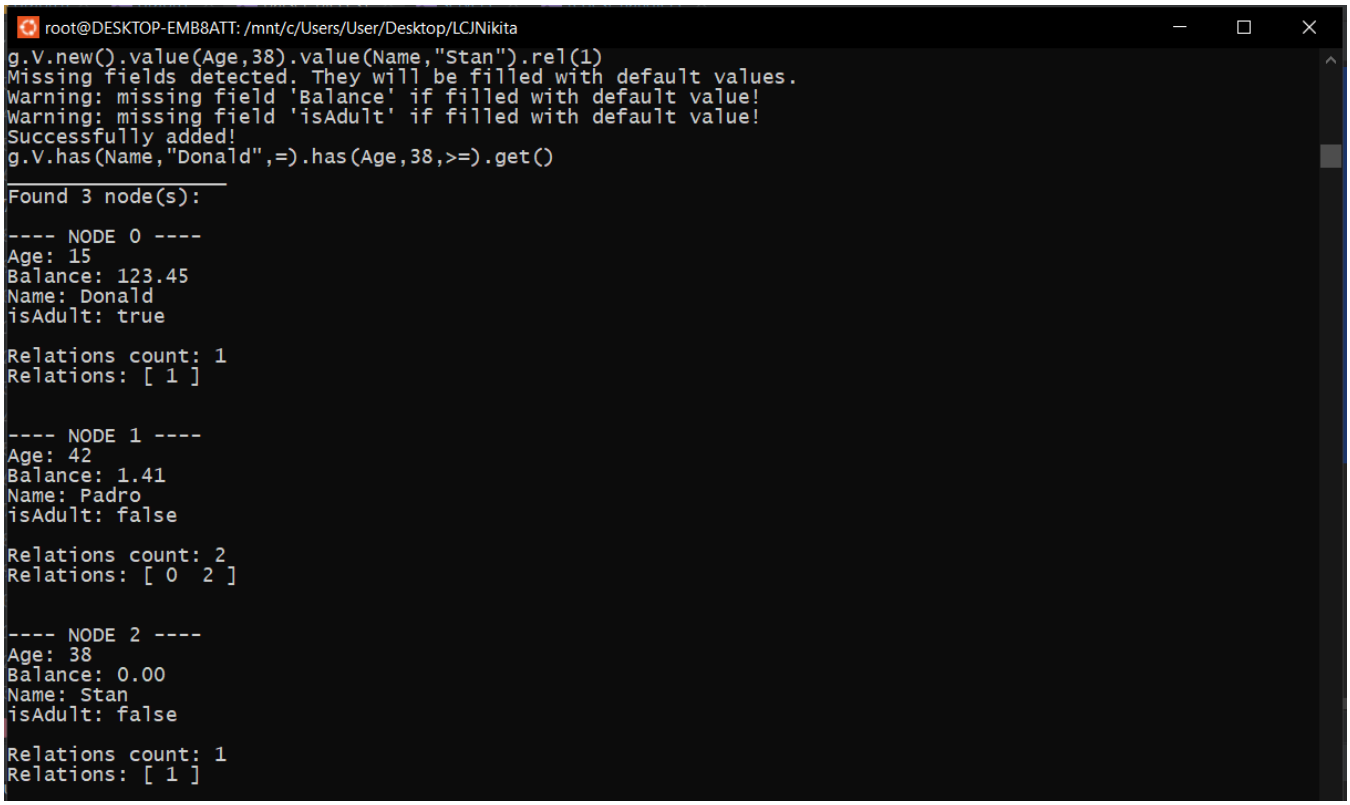




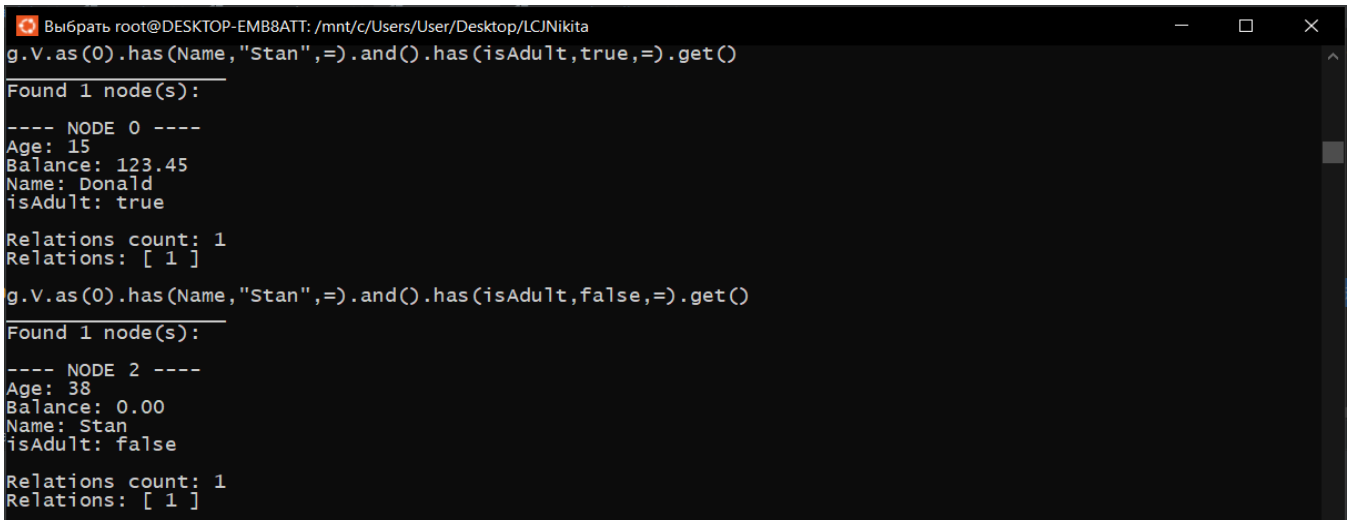
- Добавление нового элемента *со связью с предыдущим* и его выборка

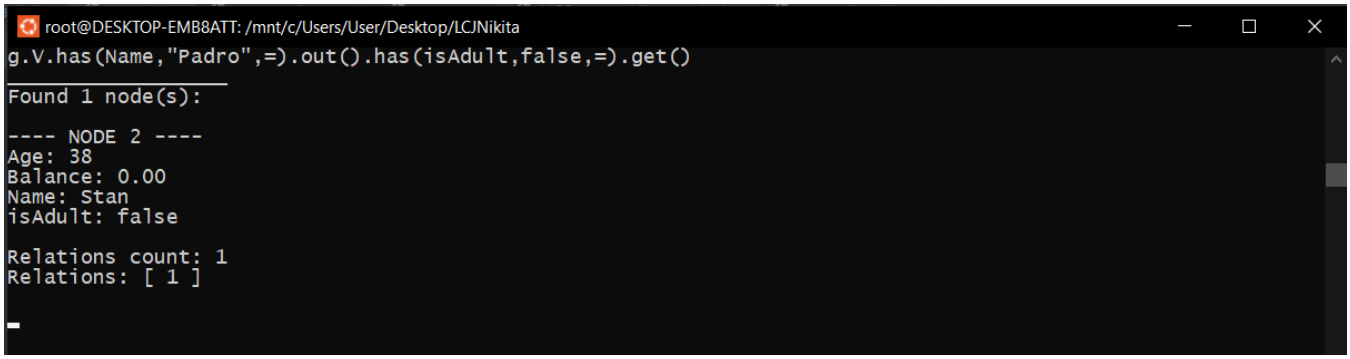


- Добавление нового элемента (с указанием не всех полей) *другим способом связывания* и его выборка

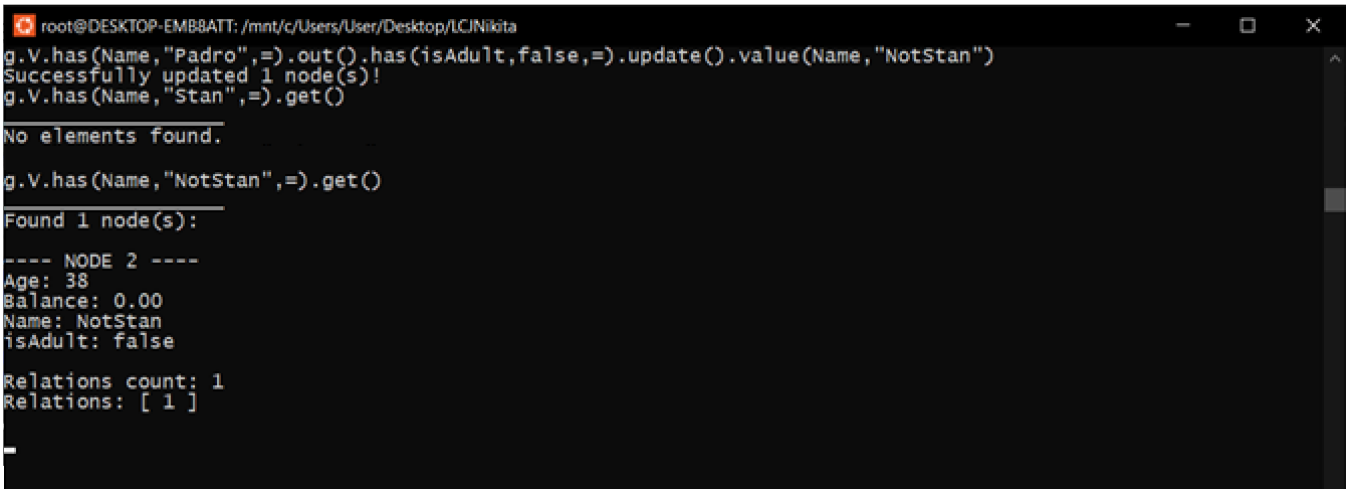


- Демонстрация работы комбинаций булевских выражений

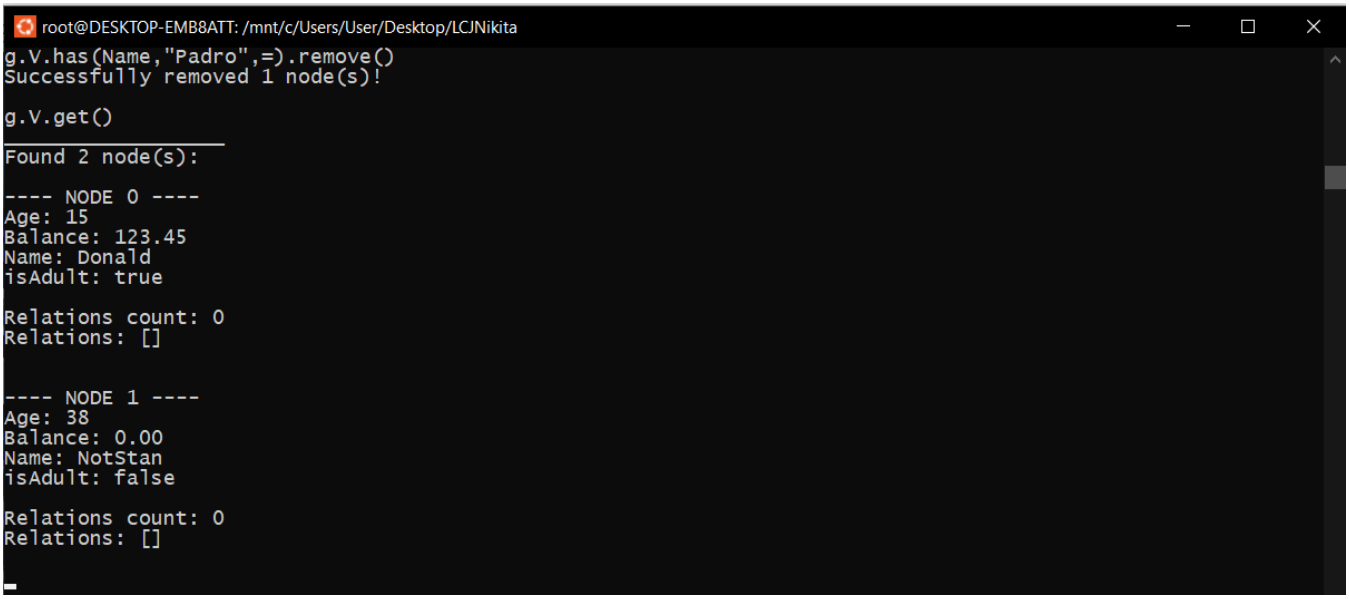




- Обновление элемента



- Удаление элементов



Выводы:

По итогам выполнения работы я научился реализовывать сетевую передачу данных на низком уровне, что, кажется, не является прямой задачей языка Си. Научился работать с файлами на байтовом уровне, взял в привычку вовремя закрывать все открытые ресурсы. А также познакомился с одним из способов сериализации данных – Protocol Buffers, который показался мне не совсем удобным ввиду того, что дерево запроса, генерируемое файлом .proto, сигнатурно отличается от описанного мной в модуле с парсингом, что заставило писать довольно ёмкий конвертер из одного дерева в другое, хотя логически они абсолютно одинаковые.