Национальный исследовательский университет ИТМО

Факультет ПИиКТ

Лабораторная работа №3 по дисциплине «Низкоуровневое программирование»

Вариант: 4 (Thrift)

Работу выполнил:

Асташин С. С.

Группа:

P33302

Преподаватель:

Кореньков Ю. Д.

Санкт-Петербург,

2023

Цель работы

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.

Задачи

1. Изучить выбранную библиотеку
2. На основе существующей библиотеки реализовать модуль, обеспечивающий взаимодействие
    a. Описать схему протокола в поддерживаемом библиотекой формате
    b. Подключить библиотеку к проекту и сформировать публичный интерфейс модуля с использованием встроенных или сгенерированных структур данных используемой библиотеки
    c. Реализовать публичный интерфейс посредством библиотеки в соответствии с п1
3. Реализовать серверную часть в виде консольного приложения
4. Реализовать клиентскую часть в виде консольного приложения

Исходный код

https://github.com/Gramdel/llp_lab3

Описание работы

Список модулей:

- client – вторая лабораторная работа, client.c и serializer.c (клиентский модуль)
- db – первая лабораторная работа
- server – server.c и deserializer.c (серверный модуль)
- structs.thrift – описание схемы протокола

Для того, чтобы начать работу с Apache Thrift, необходим structs.thrift:

```
union value_t {
    1: i32 intVal;
    2: double doubleVal;
    3: bool boolVal;
    4: string strVal;
}

struct astNode_t {
    1: optional list<astNode_t> left;
    2: optional list<astNode_t> right;
    3: nodeType_t type;
    4: value_t val;
}

service ZgdbService {
        string execute(1: astNode_t tree);
}
```

structs.thrift (в целях экономии места опущен enum)

После генерации кода получаем четыре файла: structs_types.c / structs_types.h, zgdb_service.c / zgdb_service.h.

Клиентский модуль работает так: flex и bison парсят ввод, создают дерево разбора; потом каждый узел дерево преобразуется к *astNode_t* функцией *serialize*, и преобразованное дерево отправляется на сервер функцией *zgdb_service_client_execute*.

```
while (true) {
    yyset_lineno(1); // сброс счётчика строк лексера
    yyrestart(stdin); // чистка буфера (чтобы после ошибок всё было ок)
    if (yyparse(&tree) == 0) {
        // Выход по слову "exit":
        if (!tree) {
            printf("Leaving...\n");
            break;
        }
        // Отправка запроса:
        printf("Sending request...\n");
        if (zgdb_service_client_execute(client, &response, serialize(NULL, tree), &error)) {
            printf("Response:\n%s", response);
            g_free(response);
            response = NULL;
        } else {
            g_printerr("Error: %s (code %d)\n", error->message, error->code);
            g_clear_error(&error);
            goto exit;
        }
        destroyNode(tree);
    }
}
```

client.c

Серверный модуль реализует интерфейсы, описанные в zgdb_service.c. В функции *zgdb_service_handler_impl_execute* полученное дерево преобразуется к *query* из первой лабораторной работы, после чего запрос выполняется, а его результат возвращается клиенту:

```
query* q = deserializeQueryNode(queryNode);

switch (tree->type) {
    case NODE_TYPE_T_SELECT_QUERY_NODE: {
        iterator* it;
        g_string_append(result, executeSelect(file, &errorOccurred, &it, q) ? "Successful SELECT!\n"
                                                                            : "Failed to SELECT!\n");
        while (hasNext(it)) {
            document* doc = next(file, it);
            GString* printedDoc = printDocument(doc);
            g_string_append(result, printedDoc->str);
            g_string_free(printedDoc, true);
            destroyDocument(doc);
        }
        destroyIterator(it);
        break;
    }
    case NODE_TYPE_T_INSERT_QUERY_NODE:
        g_string_append(result, executeInsert(file, &errorOccurred, q) ? "Successful INSERT!\n"
                                                                       : "Failed to INSERT!\n");
        break;
    case NODE_TYPE_T_UPDATE_QUERY_NODE:
        g_string_append(result, executeUpdate(file, &errorOccurred, q) ? "Successful UPDATE!\n"
                                                                       : "Failed to UPDATE!\n");
        break;
    case NODE_TYPE_T_DELETE_QUERY_NODE:
        g_string_append(result, executeDelete(file, &errorOccurred, q) ? "Successful DELETE!\n"
                                                                       : "Failed to DELETE!\n");
        break;
}
```

server.c

Аспекты реализации

Как оказалось, thrift_c_glib не умеет генерировать рекурсивные структуры. Изначально *astNode_t* выглядел так:

```
struct astNode_t {
    1: optional astNode_t left;
    2: optional astNode_t right;
    3: nodeType_t type;
    4: value_t val;
}
```

Файлы-то Thrift сгенерировал, только скомпилировать их не удалось из-за вот такого момента:

```
/* struct astNode_t */
struct _astNode_t
{
  ThriftStruct parent;

  /* public */
  astNode_t * left;
  gboolean __isset_left;
  astNode_t * right;
  gboolean __isset_right;
  nodeType_t type;
  gboolean __isset_type;
  value_t * val;
  gboolean __isset_val;
};
typedef struct _astNode_t astNode_t;
```

Если переместить *typedef* наверх, то компилируется код нормально, но тогда функция-конструктор для *astNode_t*, вместо заполнения *left* и *right* null, пытается их рекурсивно инициализировать, что приводит к бесконечной рекурсии. Единственное более-менее адекватное решение в данной ситуации – использовать списки (*list*) из одного элемента, поскольку такую рекурсию библиотека поддерживает.

Результаты

Ниже представлен возможный сеанс клиента:

```
user@llp-ubuntu:~/Desktop/1/llp_lab3$ ./client_exec localhost 9090
Welcome to ZGDB Client!
Connecting to localhost:9090...
Successfully connected!
insert {
    root(values: [{rootInt1 : 111}]) {
        child(values: [{str1 : "like"}, {str2 : "sssslike"}, {int1: 1000}]),
        child(values: [{str1 : "like"}, {str2 : "sssslik"}, {int1: -1000}])
    }
}
Sending request...
Response:
Successful INSERT!
select { root, root { child } }
Sending request...
Response:
Successful SELECT!
root#642A993E0000000000015FA7 {
        rootInt1 = 111
}
Successful SELECT!
child#642A993E0000000000016057 {
        str1 = "like"
        str2 = "sssslik"
        int1 = -1000
}
child#642A993E0000000000015FE6 {
        str1 = "like"
        str2 = "sssslike"
        int1 = 1000
}
update {
    root {
        child (values: [{str2 : "THAT'S WHAT I LIKE!"}], filter: like(str2, str1))
    }
}
Sending request...
Response:
Successful UPDATE!
select { root { child } }
Sending request...
Response:
Successful SELECT!
child#642A993E0000000000016057 {
        str1 = "like"
        str2 = "sssslik"
        int1 = -1000
}
child#642A993E0000000000015FE6 {
        str1 = "like"
        str2 = "THAT'S WHAT I LIKE!"
        int1 = 1000
}
delete { root }
Sending request...
Response:
Successful DELETE!
selct { root }
Error on line 1: syntax error
select { root }
Sending request...
Response:
Failed to SELECT!
exit
Leaving...
```

Выводы

В ходе выполнения данной лабораторной работы была изучена (на базовом уровне) библиотека Apache Thrift и сопутствующая ей (в случае языка C) библиотека GLib. С помощью Thrift были написаны клиентский и серверный модули для работы с “базой данных” из первой лабораторной работы. К сожалению, это всё прошло далеко не гладко: от проблем с установкой, для которой требуются не только те зависимости, которые указаны на сайте Apache, до проблем с генерацией рекурсивных структур.
Также, для успешной сборки проекта пришлось немножко подтянуть знания по CMake, поскольку собирать это дело на обычных Makefile’ах – боль.