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 Buffer)

Выполнил:
Степанов М.А.
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 из себя синтез артефактов полученных в 1 и 2 лабораторных работах (database_module), (parser_module), в частности объединение их в две взаимодействующих программы: клиент и сервер. Makefile’ы лабораторных работ были переделаны в файлы cmake и включены в новый модуль в качестве подключаемых библиотек. Для сериализации данных использовался Protocol Buffer, реализуемый библиотекой nanopb (использовалась именно она, потому что было найдено много материалов в интернете + она потребляет меньше оперативной памяти). Сборка конечных артефактов (client, server) осуществляется при помощи cmake. В качестве аргументов серверу передается адрес локальной конечной точки подключения и адрес подключаемого файла, а также имя файла, с которым будет взаимодействовать сервер. В качестве аргументов клиенту передается адрес локальной конечной точки для подключения.

Аспекты реализации:

Для работы с protocol buffer необходимо сформировать два файла (message.pb.h, message.pb.c). Для этого необходимо написать файла с расширением .proto в качестве описания протокола.

Message.proto:

```
syntax = "proto2";

message Attribute {
 required int32 type = 1;
 required string name = 2;
 oneof values {
  string str = 3;
  int32 integer = 4;
  float real = 5;
  int32 int32ean = 6;
 }
}

message Operator {
 required int32 field = 1;
 required Attribute value = 2;
}

message Comparator {
 required int32 negative = 1;
 required int32 is_true = 2;
 required int32 operation = 3;
 required Operator op1 = 4;
 required Operator op2 = 5;
}

message Filter {
 required int32 negative = 1;
 repeated Comparator comp = 2;
}

message Level {
 required int32 negative = 1;
 required int32 any = 2;
 required int32 parent = 3;
 required int32 id = 4;
 repeated Filter filters = 5;
}

message Query_tree {
 required int32 command = 1;
 repeated Level level = 2;
 required string name = 3;
 required int64 value = 4;
}

message Return_code {
 required int32 code = 1;
}
```

```
message Entity {
  repeated Attribute values = 1;
}

message Tuple {
  required Entity data = 1;
  required int32 id = 2;
  required int32 parent_id = 3;
}

message Collection {
  repeated Tuple tuples = 1;
}
```

В качестве результата работы сервер отправляет либо сущность Return_code (содержит только код возврата), либо список искомых элементов (Collection). Ответ отправляется пакетами по 1024 байт, имеется ограничение на длину названия аттрибутов в 16 символов (данное ограничение указано в конфигурации и можно изменить переконфигурирова proto файл. Так как Protocol Buffer имеет обратную совместимость, это не должно привести к устареванию данных).

Библиотека не поддерживает удаленный вызов функций, поэтому для связи по сети были использована API ОС.

Client:

```
while(1) {
    sockfd = do_connection(argv[1], hints);
    printf("client: print your request:\n");
    code = execute(sockfd);
    if (!code) {
        printf("client: successfully executed\n");
    } else {
        printf("client: error %ld was occurred\n", code);
        break;
    }
    close(sockfd);
}
```

Server:

```
while (1) {
    sin_size = sizeof their_addr;
    new_fd = accept(sockfd, (struct sockaddr *) &their_addr, &sin_size);
    if (new_fd == -1) continue;

    inet_ntop(their_addr.ss_family,get_in_addr((struct sockaddr *) &their_addr),s, sizeof s);
    printf("server: got connection from %s\n", s);

    if (!fork()) {
        close(sockfd);
        do_connection(new_fd, argv[2]);
        close(new_fd);
        exit(1);
    }
```

```
    close(new_fd);
}
```

client.do_connection:

```
for(p = serv_info; p != NULL; p = p->ai_next) {
    if ((sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1) {
        perror("client: socket\n");
        continue;
    }

    if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
        close(sockfd);
        perror("client: connect\n");
        continue;
    }

    break;
}
```

server connection:

```
if (listen(sockfd, BACKLOG) == -1) {
    perror("server: listen\n");
    exit(1);
}
```

client decode and encode:

```
ostream = pb_ostream_from_buffer(output_buffer, sizeof(output_buffer));
pb_encode(&ostream, Query_tree_fields, &tree);

if (send(sockfd, output_buffer, BUFFER_SIZE, 0) == -1) {
    perror("client: couldn't send message\n");
    close(sockfd);
}

if ((recv(sockfd, input_buffer, BUFFER_SIZE, 0)) == -1) {
    perror("client: couldn't receive message\n");
    close(sockfd);
}
istream = pb_istream_from_buffer(input_buffer, sizeof(input_buffer));

switch (tree.command) {
    case CRUD_FIND:
        pb_decode(&istream, Return_code_fields, &collection);
        for(size_t i = 0; i < collection.tuples_count; i++){
            printf("--- TUPLE %d ---\n", collection.tuples[i].id);
            printf("name: %s\n", collection.tuples[i].data.values[0].values.str);
            printf("code: %d\n", collection.tuples[i].data.values[0].values.integer);
        }
        returnCode.code = 0;break;
    default:
        pb_decode(&istream, Return_code_fields, &returnCode); break;
}
```

server decode and encode:

```
if ((recv(new_fd, input_buffer, BUFFER_SIZE, 0)) == -1) {
    perror("server: couldn't receive message\n");
}

istream = pb_istream_from_buffer(input_buffer, sizeof(input_buffer));
pb_decode(&istream, Query_tree_fields, &tree);

struct result_list_tuple *list;

uint64_t code = parse_request(filename, tree, &list);

if (tree.command == CRUD_FIND) {
    collection = get_collection(list);
    ostream = pb_ostream_from_buffer(output_buffer, sizeof(output_buffer));
    pb_encode(&ostream, Collection_fields, &collection);
} else {
    returnCode.code = code;
    ostream = pb_ostream_from_buffer(output_buffer, sizeof(output_buffer));
    pb_encode(&ostream, Return_code_fields, &returnCode);
}

if (send(new_fd, output_buffer, BUFFER_SIZE, 0) == -1) {
    perror("server: couldn't send message\n");
}
```

Результаты:

Вывод сервера не многословен (если файл не инциализирован происходит тихая инциализация как в 1 лабе):

```
/home/mike/LLP/LowLevelPrograming3/cmake-build-debug/LLP3 127.0.0.1 simple.data
server: waiting for connections...
server: got connection from 127.0.0.1
server: got connection from 127.0.0.1
server: got connection from 127.0.0.1
```

В клиенте в зависимости от запроса разный вывод:

Прим 1:

```
client: connecting to 127.0.0.1
client: print your request:
?/1/*
--- TUPLE 4 ---
name: mike
code: 123
client: successfully executed
```

Прим 2:

```
client: connecting to 127.0.0.1
client: print your request:
+/123[]/
client: error 1 was occurred
```

Прим 3:

```
client: connecting to 127.0.0.1
client: print your request:
+/123
client: successfully executed
```

Выводы:

- Изучил, неизвестную мне до этого, библиотеку nanopb и способ передачи данных по сети — Protocol Buffer. Узнал о существующих плюсах данного способа — ускорение передачи данных, уменьшение используемой памяти и минусах — абсолютной не читаемости данных человеком (и малое количество библиотек для C — их намного больше для C++)
- Научился интегрировать модули cmake, применяя библиотеки и несколько целей сборки (гораздо удобнее, чем Makefile, как мне показалось).
- Был реализован протокол передачи данных при помощи Protocol Buffer, а также передача этих данных по сети.