

Санкт-Петербургский Национальный Исследовательский Университет
ИТМО

МФКТиУ, факультет ПИиКТ

**Лабораторная работа №3 по предмету
«Низкоуровневое программирование»**

Преподаватель: Кореньков Юрий Дмитриевич

Выполнил: Стефан Лабович

Группа: P33102

Вариант: JSON

Санкт-Петербург, 2022

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 писанного формата и последовательно выполняющая 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. Клиентская часть – в цикле получающая на стандартный ввод текст команд, извлекающая из его 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.

Выбранная библиотека для работы с JSON-ом: JANSSON

В данной библиотеке чтобы работать с JSON данными используется структура `json_t`, в нем всегда содержится тип хранимого значения и счетчик ссылок данного значения.

Модуль обеспечивающий взаимодействие между клиентом и сервером:

Данный модуль занимается преобразованием структуры полученной от клиента в JSON формат который дальше ссылается серверу. Сервер потом использует данный модуль чтобы из JSON получить структуру, которую использует чтобы определить какую операцию он должен выполнить над элементами схемы или данных. Сервер потом преобразует полученный результат в JSON и отправляет его клиенту, который опять используя данный модуль преобразует данные и выводит их пользователю. Отправка и получение данных тоже делается этим модулем.

Часть модуля которая используется клиентом:

```
json_t* query_to_json(query* q);
```

Данная функция занимается преобразованием структуры которую создает парсер и которая описанна во втором лабораторном занятии.

```
void client_communicate(int sockfd)
{
    char buff[MAX];
    char string[MAX];
    for (;;) {
        query *q = yyparse();
        json_t* t = query_to_json(q);
        char *s = json_dumps(t, JSON_INDENT(2));

        memset(buff, '\\0', MAX);
        strcpy(buff, s);
    }
}
```

```

        write(sockfd, buff, sizeof(buff));
        memset(buff, '\\0', MAX);

        do {
            read(sockfd, buff, sizeof(buff));
            string = response_to_string(buff);
            if(string[0] != '\\0') {
                printf("%s", string);
            }

            } while(buff[0] != '\\0');
        memset(buff, '\\0', MAX);

    }
}

```

Данная функция занимается реализацией обмена информации между клиентом и сервером на стороне клиента.

Часть модуля которая используется сервером:

```

query_to_execute* server_decode_json(char* s);

```

Данная функция занимается преобразованием полученной информации сначала из буфера в JSON, а потом и валидацией и парсингом данного JSON в структуру query_to_execute, которая содержит всю нужную нам информацию для следующей функции.

```

void execute_query(int connfd, database* db, FILE* f, char* buff,
query_to_execute* qte);

```

Именно эта функция выбирает и выполняет нужную операцию над элементами схемы или данных.

```

void server_communicate(int connfd, database* db, FILE* f)
{
    char buff[MAX];
    for (;;) {
        memset(buff, '\\0', MAX);

        read(connfd, buff, sizeof(buff));
        printf("%s", buff);

        query_to_execute* qte = server_decode_json(buff);
        memset(buff, '\\0', MAX);

        execute_query(connfd, db, f, buff, qte);

        memset(buff, '\\0', MAX);
        write(connfd, buff, sizeof(buff));

    }
}

```

Данная функция занимается обменом информацией между сервером и клиентом на стороне клиента.

Пример сеанса работы приложения(Схема работы модуля в конкретном случае):

На стороне клиента вводим команду для создания новой таблицы.

```
CREATE TABLE studs {id: INTEGER, name:VARCHAR(32), age: INTEGER, active:
BOOLEAN};
```

Клиент посредством модуля из второй лабораторной работы, формирует структуру запроса, и посредством модуля взаимодействия из этой лабораторной работы преобразует эту структуру в JSON, с помощью данной функции:

```
json_t* query_to_json(query* q);
```

Ниже приведен кусочек кода который занимается переводом из struct query в JSON

```
json_object_set_new(root, "query_type", json_integer(q->type));
json_object_set_new(table, "name", json_string(q->ct->name));
json_object_set_new(table, "params", params);
json_object_set(root, "table", table);
json_object_set_new(root, "return_value", json_integer(OK));
column* temp = q->ct->column;
while(temp!=NULL){
    json_t* col = json_object();
    json_object_set_new(col, "name", json_string(temp->name));
    json_object_set_new(col, "column_type", json_integer(temp->type));
    json_object_set_new(col, "column_size", json_integer(temp->size));
    json_array_append(params, col);
    temp = temp->next;
    json_decref(col);
}
```

Данные серверу передаются в таком виде.

```
{
  "query_type": 0,
  "table": {
    "name": "studs",
    "params": [
      {
        "name": "id",
        "column_type": 0,
        "column_size": 4
      },
      {
        "name": "name",
        "column_type": 3,
        "column_size": 32
      },
      {
        "name": "age",
```

```

        "column_type": 0,
        "column_size": 4
    },
    {
        "name": "active",
        "column_type": 2,
        "column_size": 1
    }
]
},
"return_value": 0
}

```

Сервер потом эти данные должен распаковать в JSON и перевести в struct query_to_execute, функция которая этим занимается .

```
query_to_execute* server_decode_json(char* s);
```

Кусочек кода который этим занимается для данного случая.

```

qte->createTableQuery = malloc(sizeof(create_table_query));
json_unpack(table_name, "s", &(qte->createTableQuery->name));
for(int i = 0; i< json_array_size(params); i++){
    json_t* col = json_array_get(params, i);
    char* col_name;
    enum data_type col_type;
    uint16_t col_size;

    json_unpack(col, "{s:s, s:i, s:i}", "name", &col_name, "column_type",
    &col_type, "column_size", &col_size);
    column* col_to_add = create_column(col_name, col_type, col_size);
    cols[i] = col_to_add;
}
qte->createTableQuery->cols = cols;
qte->createTableQuery->num_cols = json_array_size(params);

```

Затем данная структура передается следующей функции:

```
void execute_query(int connfd, database* db, FILE* f, char* buff,
query_to_execute* qte);
```

Кусочек кода для конкретного случая:

```

table* t = create_table(qte->createTableQuery->name, qte->createTableQuery-
>num_cols,qte->createTableQuery->cols);
insert_table_to_schema(t, db, f);
initialize_table_record_block(t, db, f);
json_t* resp = response_to_json(OK);
strcpy(buff, resp);
write(connfd, buff, sizeof(buff));
memset(buff, '\0', MAX);

```

Клиент получает данные в JSON и выводит их в человеко-понятном виде.

```
string = response_to_string(buff);  
printf("%s", string);
```

В конце на стандартном выводе пишет:

OK

Задание сделанное на консультации(NORTHWIND):

```
FOR x IN product  
  FOR y IN category  
    FILTER x.categoryID = y.categoryID  
    FILTER product.productID == 35 OR product.productName == 'Chai'  
    RETURN {x.productID, x.productName, y.categoryID, y.categoryName};
```

	productID	productName	categoryID	categoryName
1	1	Chai	1	Beverages
2	35	Steeleye Stout	1	Beverages

```
northwind$ MATCH (p:Product)-[r:PART_OF]-(c:Category) WHERE p.productID="35" OR  
p.productName="Chai" RETURN p.productID, p.productName, c.categoryID,  
c.categoryName
```

	p.productID	p.productName	c.categoryID	c.categoryName
1	"35"	"Steeleye Stout"	"1"	"Beverages"
2	"1"	"Chai"	"1"	"Beverages"

Started streaming 2 records in less than 1 ms and completed after 1 ms.

```
FOR x IN product  
  FOR y IN category  
    FILTER x.categoryID = y.categoryID  
    FILTER product.productID == 35 OR product.productID == 51  
    FILTER category.categoryID == 1 OR category.categoryName == 'Produce'  
    RETURN {x.productID, x.productName, y.categoryID, y.categoryName};
```

	productID	productName	categoryID	categoryName
1	51	Manjimup Dried Apples	7	Produce
2	35	Steeleye Stout	1	Beverages

northwind\$ MATCH (p:Product)-[r:PART_OF]→(c:Category) WHERE (p.productID="35" OR p.productName="Manjimup Dried Apples") AND (c.categoryID="1" OR c.categoryName = "Produce") RETURN p.productID, p.productName, c.categoryID, c.categoryName

	p.productID	p.productName	c.categoryID	c.categoryName
1	"35"	"Steeleye Stout"	"1"	"Beverages"
2	"51"	"Manjimup Dried Apples"	"7"	"Produce"

Started streaming 2 records in less than 1 ms and completed after 1 ms.

```

FOR x IN product
  FOR y IN category
    FILTER x.categoryID = y.categoryID
    FILTER category.categoryID == 7 OR category.categoryName == 'Produce'
    RETURN {x.productID, x.productName, y.categoryID, y.categoryName};

```

productID	productName	categoryID	categoryName
7	Uncle Bobs Organic Dried Pears	7	Produce
14	Tofu	7	Produce
28	Rössle Sauerkraut	7	Produce
51	Manjimup Dried Apples	7	Produce
74	Longlife Tofu	7	Produce

northwind\$ MATCH (p:Product)-[r:PART_OF]→(c:Category) WHERE c.categoryID="7" AND c.categoryName="Produce" RETURN p.productID, p.productName, c.categoryID, c.categoryName

	p.productID	p.productName	c.categoryID	c.categoryName
1	"51"	"Manjimup Dried Apples"	"7"	"Produce"
2	"74"	"Longlife Tofu"	"7"	"Produce"
3	"28"	"Rössle Sauerkraut"	"7"	"Produce"
4	"14"	"Tofu"	"7"	"Produce"
5	"7"	"Uncle Bob's Organic Dried Pears"	"7"	"Produce"

Started streaming 5 records after 8 ms and completed after 9 ms.

```

FOR x IN product
  FOR y IN category
    FILTER x.categoryID = y.categoryID
    FILTER category.categoryID == 8 OR category.categoryName == 'Produce'
    FILTER product.productID == 51 OR product.productID == 35 AND category.categoryID == 7
    RETURN {x.productID, x.productName, y.categoryID, y.categoryName};

```

productID	productName	categoryID	categoryName
51	Manjimup Dried Apples	7	Produce

northwind\$ MATCH (p:Product)-[r:PART_OF]→(c:Category) WHERE (c.categoryID="1" OR c.categoryName=...

	p.productID	p.productName	c.categoryID	c.categoryName
1	"51"	"Manjimup Dried Apples"	"7"	"Produce"

Started streaming 1 records in less than 1 ms and completed after 1 ms|

```

stele@nashina: $ sudo tcpdump -i any -c 5 host 127.0.0.1
tcpdump: data link type LINUX_SLL2
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
00:29:51.972635 lo In IP localhost.46158 > localhost.9880: Flags [P.], seq 2823407802:2823408826, ack 1587112202, win 512, options [nop,nop,TS val 458155457 ecr 458131177], length 1024
00:29:51.972645 lo In IP localhost.9880 > localhost.46158: Flags [.], ack 1024, win 504, options [nop,nop,TS val 458155457 ecr 458155457], length 0
00:29:51.972750 lo In IP localhost.9880 > localhost.46158: Flags [P.], seq 1:1025, ack 1024, win 512, options [nop,nop,TS val 458155457 ecr 458155457], length 1024
00:29:51.972756 lo In IP localhost.46158 > localhost.9880: Flags [.], ack 1025, win 504, options [nop,nop,TS val 458155457 ecr 458155457], length 0
00:29:51.972761 lo In IP localhost.9880 > localhost.46158: Flags [P.], seq 1025:2049, ack 1024, win 512, options [nop,nop,TS val 458155457 ecr 458155457], length 1024
5 packets captured
12 packets received by filter
0 packets dropped by kernel

```

Выводы:

Во время выполнения данной лабораторной работы, мне удалось создать модуль обеспечивающий взаимодействие между клиентом и сервером, т.е. я успел связать первую и вторую лабораторную работу с помощью этого модуля. Также я получил опыт работы с библиотекой Jansson.