Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования
"Национальный Исследовательский Университет ИТМО"

Факультет Программной Инженерии и Компьютерных Технологий

Лабораторная работа №3

по дисциплине

«Низкоуровневое программирование»

Вариант “XML”

Выполнил:
Студент группы P33302
Тюрин Святослав Вячеславович

Преподаватель
Кореньков Юрий Дмитриевич

Санкт Петербург
2023

Задание

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.

1 Изучить выбранную библиотеку

a. Библиотека должна обеспечивать сериализацию и десериализацию с валидацией в соответствии со схемой

b. Предпочтителен выбор библиотек, поддерживающих кодогенерацию на основе схемы

c. Библиотека может поддерживать передачу данных посредством TCP соединения

- Иначе, использовать сетевые сокеты посредством API ОС

d. Библиотека может обеспечивать диспетчеризацию удалённых вызовов

- Иначе, реализовать диспетчеризацию вызовов на основе информации о виде команды

2 На основе существующей библиотеки реализовать модуль, обеспечивающий взаимодействие

a. Описать схему протокола в поддерживаемом библиотекой формате

- Описание должно включать информацию о командах, их аргументах и результатах
- Схема может включать дополнительные сущности (например, для итератора)

b. Подключить библиотеку к проекту и сформировать публичный интерфейс модуля с использованием встроенных или сгенерированных структур данных используемой библиотеки

- Поддержать установление соединения, отправку команд и получение их результатов
- Поддержать приём входящих соединений, приём команд и отправку их результатов

c. Реализовать публичный интерфейс посредством библиотеки в соответствии с п1

3 Реализовать серверную часть в виде консольного приложения

a. В качестве аргументов командной строки приложение принимает:

- Адрес локальной конечной точки для прослушивания входящих соединений
- Имя файла данных, который необходимо открыть, если он существует, иначе создать

b. Работает с файлом данных посредством модуля из задания 1

c. Принимает входящие соединения и взаимодействует с клиентами посредством модуля из п2

d. Поступающая информация о запрашиваемых операциях преобразуется из структур данных модуля взаимодействия к структурам данных модуля управления данными и наоборот

4 Реализовать клиентскую часть в виде консольного приложения

a. В качестве аргументов командной строки приложение принимает адрес конечной точки для подключения

b. Подключается к серверу и взаимодействует с ним посредством модуля из п2

c. Читает со стандартного ввода текст команд и анализирует их посредством модуля из задания 2

d. Преобразует результат разбора команды к структурам данных модуля из п2, передаёт их для обработки на сервер, возвращаемые результаты выводит в стандартный поток вывода

5 Результаты тестирования представить в виде отчёта, в который включить:

d. В части 3 привести пример сеанса работы разработанных программ

e. В части 4 описать решение, реализованное в соответствии с пп.2-4

f. В часть 5 включить составленную схему п.2a

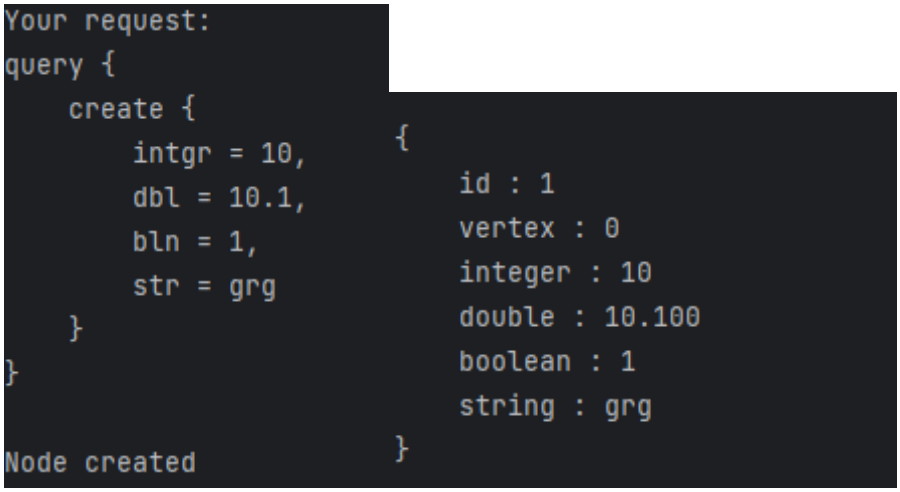
Ход работы

- Клиент получает запрос, парсит его в структуру из прошлой лабораторной request, затем упаковывает эту структуру в XML и отправляет запрос на сервер. Получает ответ и выводит его на экран.
- Сервер в свою очередь получает запрос, определяется с типом запроса, дергает нужную crud функцию, получает результат (опционально) и формирует ответ, затем отправляет.

Пример работы программы

Слева окно клиента, справа сервера.

Добавление node



Удаление node по id

Удаление всех nodes

```
Your request:
query {
    delete {
        id = *
    }
}

Nodes deleted
```

Поиск node по id

```
Your request:
query {
    find {
        id = 1
    }
}

{
        id : 1
        neighbour's id's :
        integer : 10
        double : 10.10
        boolean : 1
        string : grg
}
```

```
{
    id : 1
    vertex : 0
    integer : 10
    double : 10.100
    boolean : 1
    string : grg
}
{
    id : 2
    vertex : 0
    integer : 10
    double : 10.100
    boolean : 1
    string : grg
}
```

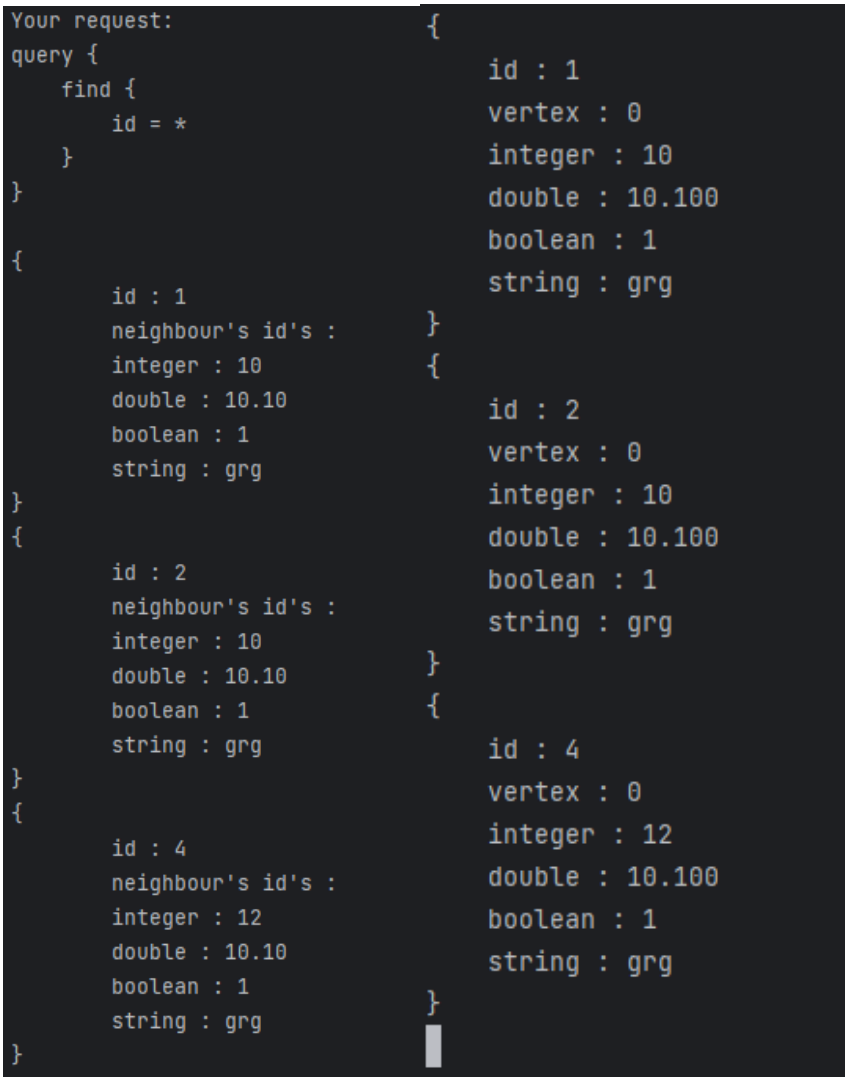
Поиск всех nodes, удовлетворяющих условию

```
Your request:
query {
    find {
        intgr = 10
    }
}

{
        id : 1
        neighbour's id's :
        integer : 10
        double : 10.10
        boolean : 1
        string : grg
}
{
        id : 2
        neighbour's id's :
        integer : 10
        double : 10.10
        boolean : 1
        string : grg
}
```

```
{
    id : 1
    vertex : 0
    integer : 10
    double : 10.100
    boolean : 1
    string : grg
}
{
    id : 2
    vertex : 0
    integer : 10
    double : 10.100
    boolean : 1
    string : grg
}
{
    id : 4
    vertex : 0
    integer : 12
    double : 10.100
    boolean : 1
    string : grg
}
```

Поиск всех node

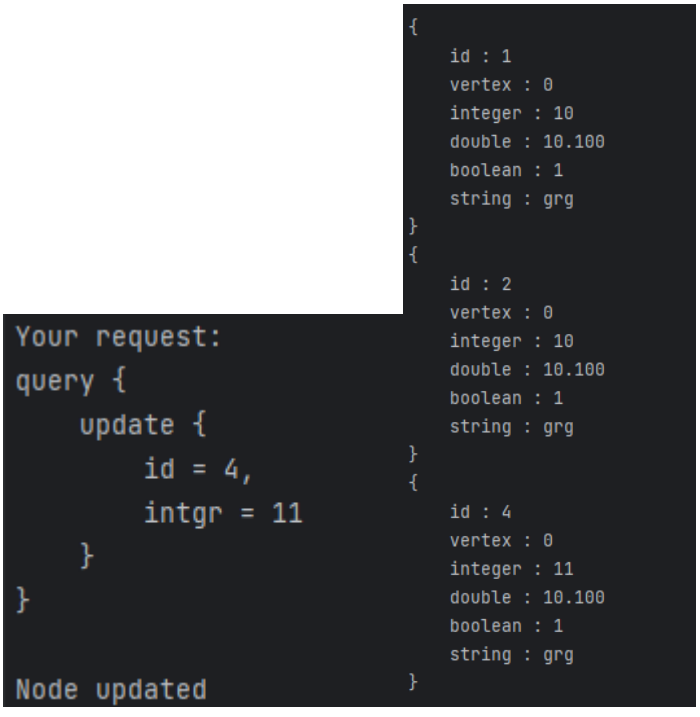


```
Your request:
query {
    find {
        id = *
    }
}

{
        id : 1
        neighbour's id's :
        integer : 10
        double : 10.10
        boolean : 1
        string : grg
}
{
        id : 2
        neighbour's id's :
        integer : 10
        double : 10.10
        boolean : 1
        string : grg
}
{
        id : 4
        neighbour's id's :
        integer : 12
        double : 10.10
        boolean : 1
        string : grg
}
```

```
{
    id : 1
    vertex : 0
    integer : 10
    double : 10.100
    boolean : 1
    string : grg
}
{
    id : 2
    vertex : 0
    integer : 10
    double : 10.100
    boolean : 1
    string : grg
}
{
    id : 4
    vertex : 0
    integer : 12
    double : 10.100
    boolean : 1
    string : grg
}
```

Обновление поля node по id



```
Your request:
query {
    update {
        id = 4,
        intgr = 11
    }
}

Node updated
```

```
{
    id : 1
    vertex : 0
    integer : 10
    double : 10.100
    boolean : 1
    string : grg
}
{
    id : 2
    vertex : 0
    integer : 10
    double : 10.100
    boolean : 1
    string : grg
}
{
    id : 4
    vertex : 0
    integer : 11
    double : 10.100
    boolean : 1
    string : grg
}
```

Присоединение двух nodes по id

```
Your request:
query {
    connect {
        id_1 = 1,
        id_2 = 2
    }
}

Node connected
```

```
{
        id : 1
        neighbour's id's : 2
        integer : 10
        double : 10.10
        boolean : 1
        string : grg
}
{
        id : 2
        neighbour's id's : 1
        integer : 10
        double : 10.10
        boolean : 1
        string : grg
}
```

Аспекты реализации

Как выглядит упакованный запрос в XML.

```
Your request:
query {
    delete {
        id = 5
    }
}

<?xml version="1.0"?>
<delete><node id="5" operand_1="="/></delete>

Not successful
```

```
Your request:
query {
    create {
        intgr = 10,
        dbl = 10.1,
        bln = 1,
        str = grg
    }
}

<?xml version="1.0"?>
<create><node intgr="10" operand_1="=" dbl="10.1" operand_2="=" bln="1" operand_3="=" str="grg" operand_4="="/></create>

Node created
```

Передача по сети происходит через API ОС.

```c
int Socket(int domain, int type, int protocol) {
    int res = socket(domain, type, protocol);
    if (res == -1) {
        perror( s: "socket failed");
        exit( status: EXIT_FAILURE);
    }
    return res;
}

void Bind(int sock_fd, const struct sockaddr *addr, socklen_t addr_len) {
    int res = bind( fd: sock_fd, addr,  len: addr_len);
    if (res == -1) {
        perror( s: "bind failed");
        exit( status: EXIT_FAILURE);
    }
}

void Listen(int sock_fd, int back_log) {
    int res = listen( fd: sock_fd,  n: back_log);
    if (res == -1) {
        perror( s: "listen failed");
        exit( status: EXIT_FAILURE);
    }
}

int Accept(int sock_fd, struct sockaddr *addr, socklen_t *addr_len) {
    int res = accept( fd: sock_fd, addr, addr_len);
    if (res == -1) {
        perror( s: "accept failed");
        exit( status: EXIT_FAILURE);
    }
    return res;
}
void Connect(int sock_fd, const struct sockaddr *addr, socklen_t addr_len) {
    int res = connect( fd: sock_fd, addr,  len: addr_len);
    if (res == -1) {
        perror( s: "connect failed");
        exit( status: EXIT_FAILURE);
    }
}

void Inet_pton(int af, const char *src, void *dst) {
    int res = inet_pton(af,  cp: src,  buf: dst);
    if (res == 0) {
        printf( format: "inet error\n");
        exit( status: EXIT_FAILURE);
    }
    if (res == -1) {
        perror( s: "inet failed");
        exit( status: EXIT_FAILURE);
    }
}
```

Сервер

```c
int start_server(int port) {
    struct sockaddr_in adr = {0};
    int server = Socket( domain: AF_INET,  type: SOCK_STREAM,  protocol: 0);
    adr.sin_family = AF_INET;
    adr.sin_port = htons( hostshort: port);
    Bind( sock_fd: server,  addr: (struct sockaddr *) &adr,  addr_len: sizeof adr);
    return server;
}

int handler_request(int server, char buf[]) {
    struct sockaddr_in adr = {0};
    Listen( sock_fd: server,  backlog: 5);
    socklen_t adr_len = sizeof adr;
    int fd = Accept( sock_fd: server,  addr: (struct sockaddr *) &adr,  addr_len: &adr_len);
    ssize_t n_read;
    n_read = read(fd, buf,  nbytes: MAX_RESPONSE_SIZE);
    if (n_read == -1) {
        perror( s: "read failed");
        exit( status: EXIT_FAILURE);
    }
    if (n_read == 0) {
        printf( format: "nothing question mb error\n");
    }
    return fd;
}

void send_response(char *msg, int fd) {
    write(fd,  buf: msg,  n: MAX_RESPONSE_SIZE);
}

void finish_server(int server) {
    close( fd: server);
}
```

Клиент

```c
void sendRequest(int port, int str_len, char request[]) {
    int fd = Socket( domain: AF_INET,  type: SOCK_STREAM,  protocol: 0);
    struct sockaddr_in adr = {0};
    adr.sin_family = AF_INET;
    adr.sin_port = htons( hostshort: port);
    Inet_pton( af: AF_INET,  src: "127.0.0.1",  dst: &adr.sin_addr);
    Connect( sock_fd: fd,  addr: (struct sockaddr *) &adr,  addr_len: sizeof adr);
    write(fd,  buf: request,  n: str_len - 1);


    char buf[MAX_RESPONSE_SIZE];
    ssize_t n_read;

    n_read = read(fd, buf,  nbytes: MAX_RESPONSE_SIZE);

    if (n_read == -1) {
        perror( s: "read failed\n");
        exit( status: EXIT_FAILURE);
    }

    if (n_read == 0) {
        printf( format: "nothing answer mb error?\n");
    }

    printf( format: "%s\n", buf);

    close(fd);
}
```

Пример работы на тестовом наборе со связью, как можно заметить результат идентичный:

```
svytoq@DESKTOP-K86V65N:~/xml/client$ ./main 8080
Your request:
query {
    interact (weight = 20) {
        node {
            name
        }
        node {
            name
        }
    }
}

Your response:
{
        weight=20
        name1=Arya-Stark        name2=Serio-Forel
}
{
        weight=20
        name1=Luwin          name2=Robb-Stark
}
{
        weight=20
        name1=Mord          name2=Tyrion-Lannister
}
{
        weight=20
        name1=Rickon-Stark          name2=Robb-Stark
}
```

```
neo4j$ MATCH p=(a)-[r:INTERACTS1]→(b) WHERE r.weight =
       20 RETURN a.name, b.name, r.weight
```

Table | Text | Code

	a.name	b.name	r.weight
1	"Arya-Stark"	"Syrio-Forel"	20
2	"Luwin"	"Robb-Stark"	20
3	"Mord"	"Tyrion-Lannister"	20
4	"Rickon-Stark"	"Robb-Stark"	20

Started streaming 4 records after 4 ms and completed after 6 ms.