

Федеральное государственное автономное образовательное учреждение высшего образования  
"Национальный Исследовательский Университет ИТМО"  
Мегафакультет Компьютерных Технологий и Управления  
Факультет Программной Инженерии и Компьютерной Техники



**Лабораторная №3**  
по дисциплине  
**'Низкоуровневое программирование'**

Выполнил Студент группы Р33102  
**Лапин Алексей Александрович**  
Преподаватель:  
**Кореньков Юрий Дмитриевич**

г. Санкт-Петербург  
2023г.

# Содержание

<b>1</b>	<b>Цель:</b>	<b>3</b>
<b>2</b>	<b>Порядок выполнения:</b>	<b>3</b>
<b>3</b>	<b>Описание работы и реализации:</b>	<b>4</b>
3.1	Серверная часть . . . . .	4
3.1.1	Прием клиентов и старт сервера . . . . .	4
<b>4</b>	<b>Результаты работы программы</b>	<b>6</b>
<b>5</b>	<b>Валидация xml схемы:</b>	<b>7</b>
5.1	response.xsd . . . . .	7
5.2	request.xsd . . . . .	8
<b>6</b>	<b>Выводы:</b>	<b>12</b>

# 1 Цель:

Выданный вариант задания: XML 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 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 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.

# 2 Порядок выполнения:

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
    - Поддерживать установление соединения, отправку команд и получение их результатов
    - Поддерживать 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:

- (a) В части 3 привести пример сеанса работы разработанных программ
- (b) В части 4 описать решение, реализованное в соответствии с пп.2-4
- (c) В часть 5 включить составленную схему п.2а

### 3 Описание работы и реализации:

- Использовалась библиотека libxml2 для работы с xml.
- Сетевое взаимодействие реализовано посредством сокетов.
- Перед отправкой сообщения отправляется его длина, это справедливо и для requests, и для response.
- Ответ от сервера содержит в себе:
  - Статус. ERROR – ошибка, OK – успешно
  - Сообщение.
  - Таблица(optional).

#### 3.1 Серверная часть

- Сервер принимает на вход 2 аргумента: адрес и порт.
- После запуска сервер начинает слушать входящие соединения.
- При подключении клиента, сервер создает новый поток, в котором обрабатывает запросы клиента.
- При получении запроса, сервер десериализует его в ast, передает в модуль выполнения запроса.
- После получения ответа от модуля выполнения запроса, сервер сериализует ответ в xml и отправляет его клиенту.

##### 3.1.1 Прием клиентов и старт сервера

```
int main(int argc, char **argv) {
    char *filename = DEFAULT_FILE;
    int port = DEFAULT_PORT;
    if (argc > 1) {
        port = atoi(argv[1]);
    }
    if (argc > 2) {
```

```

    filename = argv[2];
}

db_t *db = db_init(filename);

int sock = init_socket(port);
if (sock < 0) {
    return 1;
}

if (listen_socket(sock) < 0) {
    return 1;
}

logger(LL_INFO, __func__, "Listening on port %d", port);

signal(SIGTERM, sigint_handler);

FD_ZERO(&readfds);
FD_SET(sock, &readfds);
max_sd = sock;

struct timeval timeout;
while (server_running) {
    fd_set tmpfds = readfds;
    timeout.tv_sec = 1; // Set a timeout 1 second to allow periodic checks for
                        // server_running
    timeout.tv_usec = 0;

    int activity = select(max_sd + 1, &tmpfds, NULL, NULL, &timeout);

    if ((activity < 0) && (errno != EINTR)) {
        fprintf(stderr, "select error: %s", strerror(errno));
        break;
    }

    if (activity > 0) {
        // Check if the activity is on the server socket (new connection)
        if (FD_ISSET(sock, &tmpfds)) {
            int client = accept_socket(sock);
            if (client < 0) {
                perror("accept failed");
                continue;
            }

            logger(LL_INFO, __func__, "Client %d connected", client);

            // Add new socket to the array of sockets
            FD_SET(client, &readfds);
            if (client > max_sd) {
                max_sd = client;
            }
        }
    }
}

```

```

    }

    struct handler_args *args = malloc(sizeof(struct handler_args));
    args->db = db;
    args->client = client;

    pthread_t client_thread;
    if (pthread_create(&client_thread, NULL, (void *) client_handler,
        args) != 0) {
        perror("Failed to create thread");
        server_running = false;
        continue;
    }
    pthread_detach(client_thread);
}
}

for (int i = 0; i <= max_sd; i++) {
    if (FD_ISSET(i, &readfds)) {
        if(i != -1){
            close_socket(i);
        }
    }
}
db_close();
return 0;
}

```

## 4 Результаты работы программы

```

./lab3-client 127.0.0.1 8080
> CREATE users WITH { name: string, lastname: string, student: bool, money: int,
    score: float}
Message: Table users created successfully
> CREATE group WITH { group_id: int, name: string }
Message: Table group created successfully

```

```

> INSERT { name: "Alex", lastname: "Lapin", student: true, money: 10000, score: 5.0
    } INTO users
Message: Row inserted successfully
> INSERT { name: "Berman", lastname: "Clock", student: true, money: 232, score: 3.2
    } INTO users
Message: Row inserted successfully
> INSERT { name: "Cristian", lastname: "Ronaldo", student: false, money: 100,
    score: 4.2 } INTO users
Message: Row inserted successfully
> INSERT { name: "Dima", lastname: "Koval", student: true, money: 2330, score: 2.1
    } INTO users
Message: Row inserted successfully
> INSERT { name: "Egor", lastname: "Flagman", student: true, money: 100, score: 1.2
    } INTO users

```

```

Message: Row inserted successfully
> INSERT { name: "Fedor", lastname: "Champion", student: false, money: 0, score:
5.0 } INTO users
Message: Row inserted successfully
> INSERT { group_id: 1, name: "P33102" } INTO group
Message: Row inserted successfully
> INSERT { group_id: 2, name: "M33103" } INTO group
Message: Row inserted successfully
> INSERT { group_id: 3, name: "G33104" } INTO group
Message: Row inserted successfully
> INSERT { group_id: 4, name: "Z33105" } INTO group
Message: Row inserted successfully
> INSERT { group_id: 5, name: "K33106" } INTO group
Message: Row inserted successfully

```

```

> FOR u IN users FILTER u.money > 100 return u
Message: Selected successfully
Table:
name      lastname  student  money  score
Alex      Lapin     true     10000  5.000000
Berman    Clock     true     232    3.200000
Dima      Koval     true     2330   2.100000

> FOR u IN users FILTER u.money == 100 FOR g IN group FILTER g.group_id == 3 RETURN
MERGE(u,g)
Message: Selected successfully
Table:
name      lastname  student  money  score      group_id  name
Cristian  Ronaldo  false    100    4.200000   3         G33104
Egor      Flagman  true     100    1.200000   3         G33104

```

```

> CREATE players WITH { username: string, player: bool, cash: int, score: float}
Message: Table players created successfully
> DROP players
Message: Table players dropped successfully
> FOR p in players RETURN p
Message: Failed to find table players

```

## 5 Валидация xml схемы:

### 5.1 response.xsd

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="response">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="message" type="xs:string" minOccurs="0"/>
        <xs:element name="table" minOccurs="0">
          <xs:complexType>

```

```

<xs:sequence>
  <xs:element name="schema">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="field"
          maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="name"
                type="xs:string" use="required"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="rows" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="row" maxOccurs="unbounded"
            minOccurs="0">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="element"
                    maxOccurs="unbounded">
                      <xs:complexType mixed="true">
                        <xs:attribute name="name"
                          type="xs:string"
                          use="required"/>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:element>
</xs:sequence>
<xs:attribute name="status" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:element>
</xs:schema>

```

## 5.2 request.xsd

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- definition of simple elements -->
  <xs:element name="int" type="xs:integer"/>
  <xs:element name="string" type="xs:string"/>

```



```

<xs:element name="float" type="xs:float"/>
<xs:element name="bool" type="xs:string"/>

<!-- definition of attributes -->

<!-- definition of groups -->
<xs:group name="valueChoice">
  <xs:choice>
    <xs:element ref="int"/>
    <xs:element ref="string"/>
    <xs:element ref="float"/>
    <xs:element ref="bool"/>
  </xs:choice>
</xs:group>

<!-- definition of complex elements -->
<xs:element name="attr_name">
  <xs:complexType>
    <xs:attribute name="variable" type="xs:string" use="required"/>
    <xs:attribute name="attribute" type="xs:string" />
  </xs:complexType>
</xs:element>

<xs:element name="filter_expr">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="attr_name"/>
      <xs:group ref="valueChoice"/>
    </xs:sequence>
    <xs:attribute name="cmp" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="conditions">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="filter_expr"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="filter">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="conditions"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="forSecondType">
  <xs:sequence>
    <xs:element name="list" type="forList" minOccurs="0"
      maxOccurs="unbounded"/>

```

```

    </xs:sequence>
    <xs:attribute name="var" type="xs:string" use="required"/>
    <xs:attribute name="tablename" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="forList">
    <xs:sequence>
        <xs:element ref="filter" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="for" type="forSecondType" minOccurs="0"
            maxOccurs="1"/>
    </xs:sequence>
</xs:complexType>

<xs:element name="merge">
    <xs:complexType>
        <xs:attribute name="var1" type="xs:string" use="required"/>
        <xs:attribute name="var2" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>

<xs:element name="return">
    <xs:complexType>
        <xs:choice>
            <xs:element ref="attr_name"/>
            <xs:element ref="merge"/>
        </xs:choice>
    </xs:complexType>
</xs:element>

<xs:element name="for">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="list" type="forList" minOccurs="0" />
            <xs:element ref="return" minOccurs="0" />
            <xs:element ref="remove" minOccurs="0"/>
            <xs:element ref="update" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="var" type="xs:string" use="required"/>
        <xs:attribute name="tablename" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>

<xs:element name="pair">
    <xs:complexType>
        <xs:sequence>
            <xs:group ref="valueChoice"/>
        </xs:sequence>
        <xs:attribute name="key" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>

<xs:complexType name="insertList">
    <xs:sequence>

```

```

        <xs:element ref="pair" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:element name="insert">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="list" type="insertList" minOccurs="0"
                maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="tablename" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>

<xs:element name="definition">
    <xs:complexType>
        <xs:attribute name="name" type="xs:string" use="required"/>
        <xs:attribute name="type" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>

<xs:complexType name="defList">
    <xs:sequence>
        <xs:element ref="definition" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:element name="create">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="list" type="defList" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="tablename" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>

<xs:element name="drop">
    <xs:complexType>
        <xs:attribute name="tablename" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>

<xs:element name="remove">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="attr_name"/>
        </xs:sequence>
        <xs:attribute name="tablename" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>

<xs:element name="update">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="attr_name"/>

```

```

        <xs:element name="list" type="insertList" minOccurs="0"
            maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="tablename" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>

<xs:element name="root">
    <xs:complexType>
        <xs:sequence>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element ref="for" />
                <xs:element ref="insert" />
                <xs:element ref="create" />
                <xs:element ref="drop" />
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>

```

## 6 Выводы:

Что я узнал, чему научился:

1. Реализовать поддержку xml схемы в libxml2
2. Реализовать сетевое взаимодействие с помощью сокетов
3. Работать с xml сообщениями согласно схемам
4. Сериализовывать и десериализовывать xml сообщения в структуры данных и наоборот