Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО”

Факультет ПИиКТ

ОТЧЁТ

По лабораторной работе № 3

По предмету: Низкоуровневое программирование

Вариант: XML

Студент:

Андрейченко Леонид Вадимович

Группа P33301

Преподаватель:

Кореньков Юрий Дмитриевич

Санкт – Петербург

2023

Цель

На базе данного транспортного формата описать схему протокола обмена информацией и воспользоваться существующей библиотекой по выбору для реализации модуля, обеспечивающего его функционирование.

Задачи

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.

Описание работы

По варианту, я должен передавать запросы на сервер в формате XML, в качестве библиотеки мною была выбрана libxml2. Она является основой для библиотеки libxslt, которая позволяет анализировать таблицы стилей XSLT, а ещё её функционал в реализации c++ больше и приятнее).

Основной алгоритм взаимодействия клиента и сервера

1. Запускаются клиент и сервер на одном порту, сервер начинает прослушивать данный порт и ждать ответ. Клиент просит пользователя ввести запрос.
2. Запрос клиента (строка) преобразуется в структуру запроса (реализация во 2й лабораторной)
3. Анализируя полученную структуру, создается структура xmlDocPtr, которая определена в библиотеке и хранит дерево xml моего запроса. Эта структура передается по TCP соединению серверу
4. Сервер принимает данный XML, преобразует его в структуру запроса
5. Данная структура анализируется и выполняется данный запрос
6. Сервер отправляет в качестве ответа строку, которую печатает на стандартный вывод клиент

Аспекты реализации

Из-за особенности реализации структуры дерева XML, пришлось чуть изменить запросы. Дело в том, что libxml2 не дает заменять операции в свойствах (по умолчанию всегда равно), поэтому пришлось в отдельное свойство выносить знак. Также библиотека не позволяет хранить в качестве имени ноды символ, поэтому прошлось заменить:

- + : add
- - : remove
- ? : find
- = : update
- * : all

```
Enter path:
+/2/3[age=100][name=leo]
<?xml version="1.0" encoding="UTF-8"?>
<add>
  <tuple id="2">
    <tuple id="3" age="100" oper1="=" name="leo" oper2="="/>
  </tuple>
</add>
```

Сам запрос кладется в структуру следующим образом: Самой первой нодой кладется операция, после нее – как в запросах из лабораторной 2, id тепла это имя ноды, ей атрибуты – свойства данной ноды.

Пример создания структуры xmlDocPtr

```
xmlDocPtr request_tree = xmlNewDoc(BAD_CAST "1.0");
xmlNodePtr root = xmlNewNode(NULL, BAD_CAST "add");
xmlDocSetRootElement(request_tree, root);
xmlNodePtr lastNode = request_tree->last;
xmlNodePtr curNode = xmlNewChild(lastNode, NULL, BAD_CAST "tuple", NULL);
xmlNewProp(curNode, BAD_CAST "id", BAD_CAST cur_tuple->tupleid);
```

Сервер принимает данную структуру и с помощью реализованных в библиотеке методов позволяет итерироваться по структуре и удобно с ней работать. Все отправляемые от сервера ответы – строки, которые формируется в зависимости от результатов на нем.

Передача данных по сети реализована с помощью API ОС. Пример реализации со стороны сервера.

Методы обертки, используемые при соединении

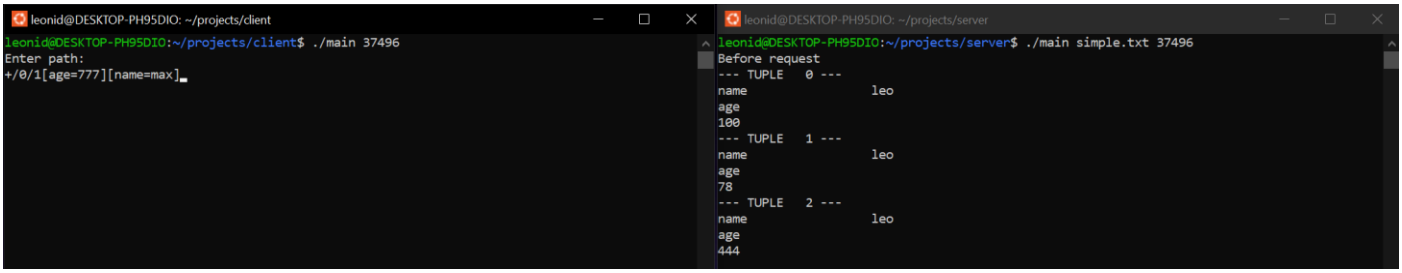
```c
int Socket(int domain, int type, int protocol) {
    int res = socket(domain, type, protocol);
    if (res == -1) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }
    return res;
}

void Bind(int sockfd, const struct sockaddr* addr, socklen_t addrlen) {
    int res = bind(sockfd, addr, addrlen);
    if (res == -1) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
}

void Listen(int sockfd, int backlog) {
    int res = listen(sockfd, backlog);
    if (res == -1) {
        perror("listen failed");
        exit(EXIT_FAILURE);
    }
}

int Accept(int sockfd, struct sockaddr* addr, socklen_t* addrlen) {
    int res = accept(sockfd, addr, addrlen);
    if (res == -1) {
        perror("accept failed");
        exit(EXIT_FAILURE);
    }
    return res;
}

void Connect(int sockfd, const struct sockaddr* addr, socklen_t addrlen) {
    int res = connect(sockfd, addr, addrlen);
    if (res == -1) {
        perror("connect failed");
        exit(EXIT_FAILURE);
    }
}

void Inet_pton(int af, const char* src, void* dst) {
    int res = inet_pton(af, src, dst);
    if (res == 0) {
        printf("inet_pton failed: src does not contain a character"
            " string representing a valid network address in the specified"
            " address family\n");
        exit(EXIT_FAILURE);
    }
    if (res == -1) {
        perror("inet_pton failed");
        exit(EXIT_FAILURE);
    }
}
```

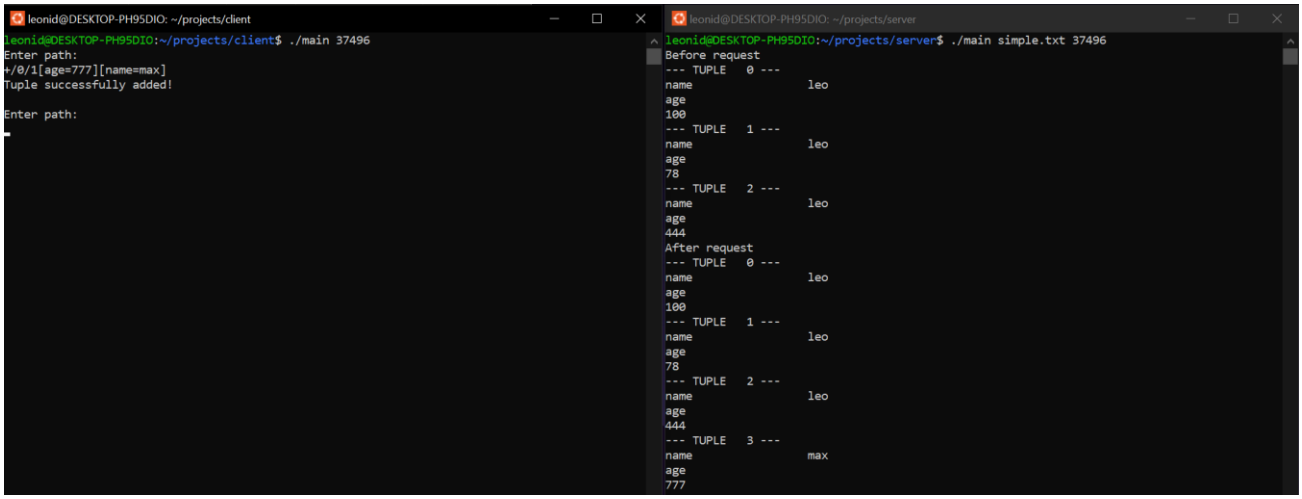
Интерфейс взаимодействия

```c
int start_server(int port) {
    struct sockaddr_in adr = { 0 };
    int server = Socket(AF_INET, SOCK_STREAM, 0);
    adr.sin_family = AF_INET;
    adr.sin_port = htons(port);
    Bind(server, (struct sockaddr*)&adr, sizeof adr);
    return server;
}

int handler_request(int server, char buf[]) {
    struct sockaddr_in adr = { 0 };
    Listen(server, 5);
    socklen_t adrlen = sizeof adr;
    int fd = Accept(server, (struct sockaddr*)&adr, &adrlen);
    ssize_t nread;
    nread = read(fd, buf, 1024);
    if (nread == -1) {
        perror("read failed");
        exit(EXIT_FAILURE);
    }
    if (nread == 0) {
        printf("END OF FILE occured\n");
    }
    return fd;
}

void send_response(char* msg, int fd) {
    write(fd, msg, 128);
}

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

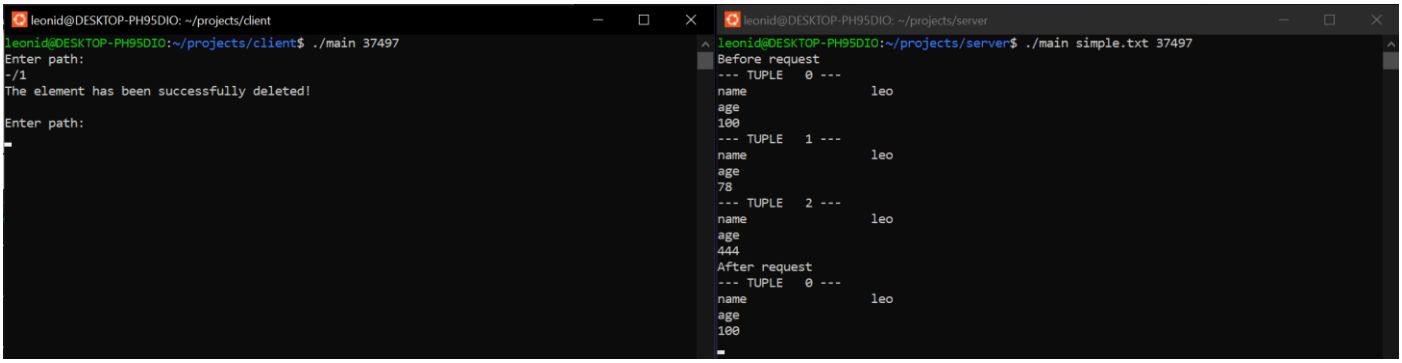
Примеры запросов

Вставка нового элемента

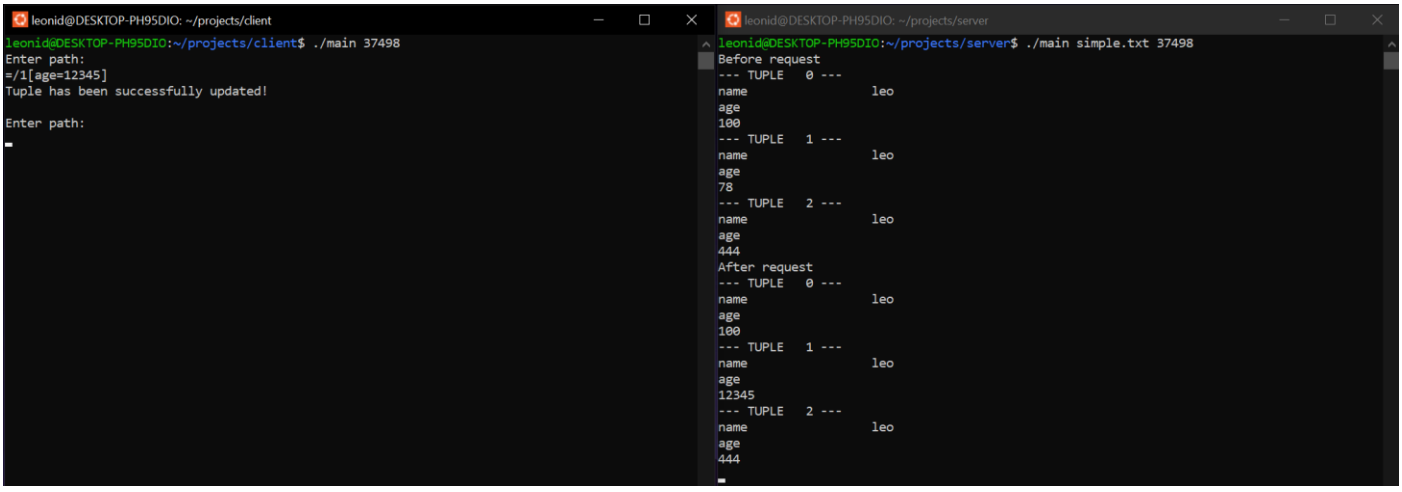




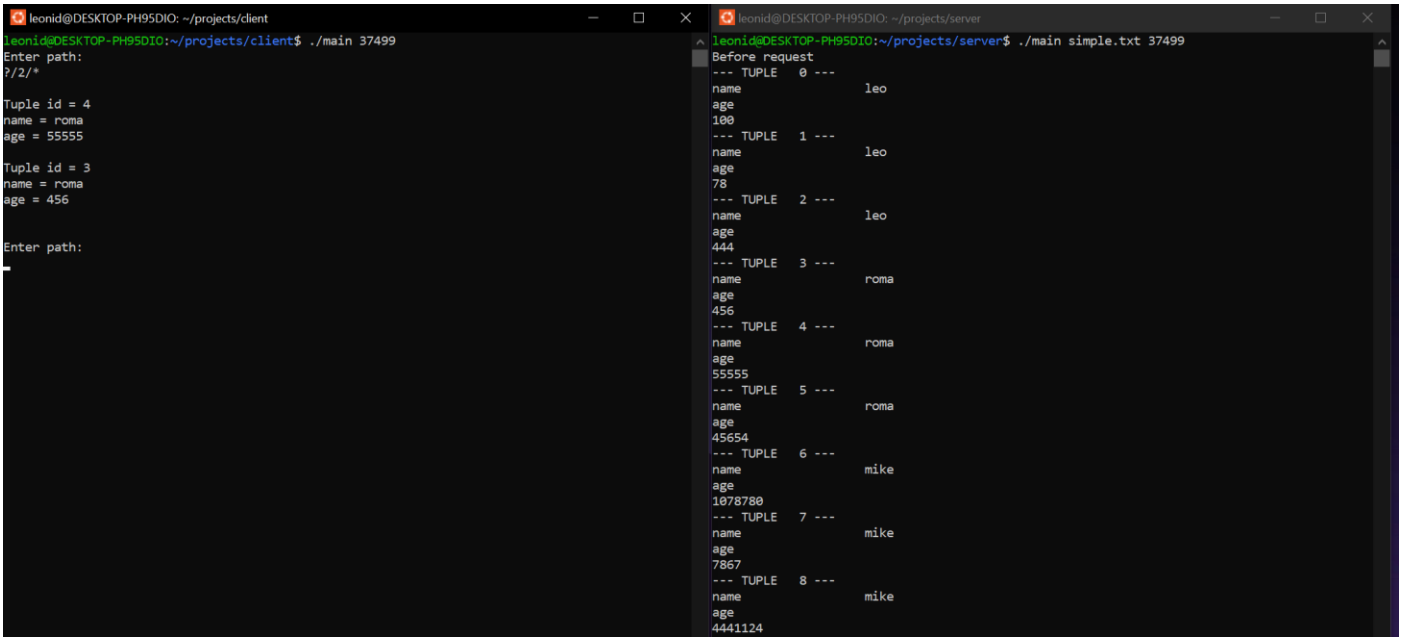
Рекурсивное удаление элементов



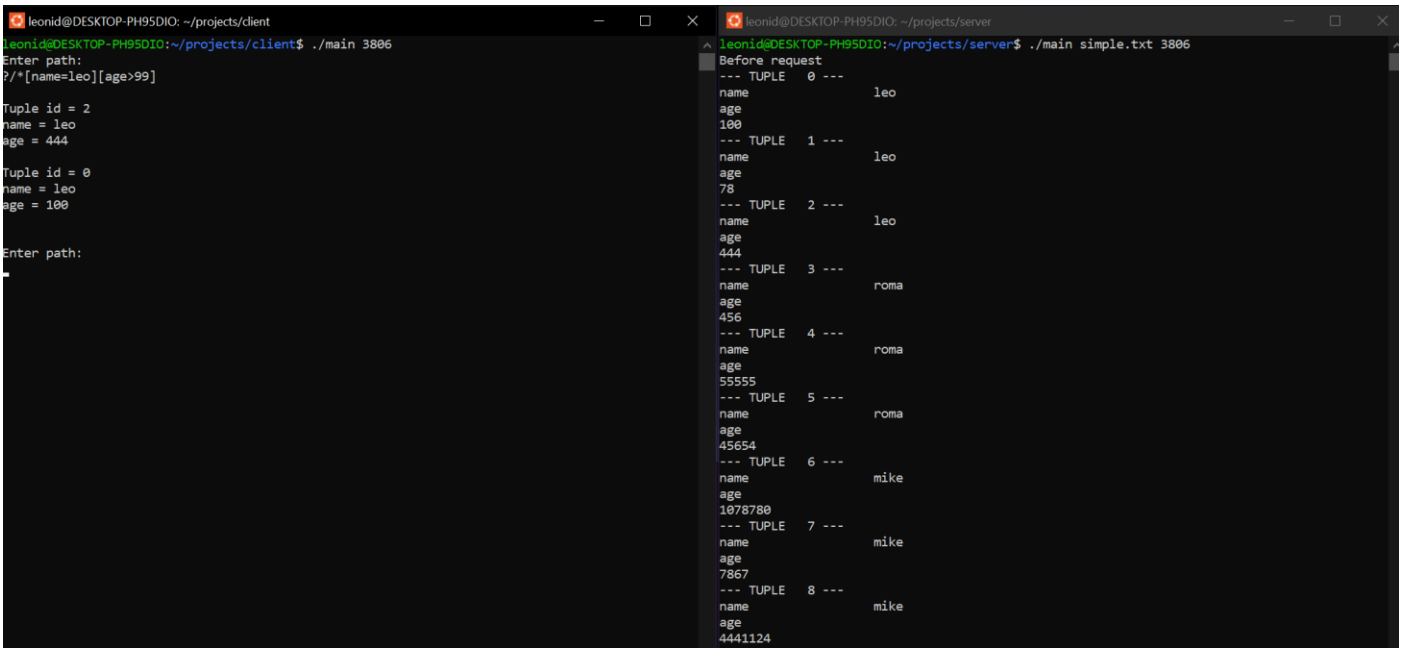
Изменение полей элемента



Рекурсивной поиск всех узлов, у которых общий предок



Запрос с атрибутами



Выводы

В ходе выполнения данной лабораторной работы я изучил библиотеку libxml2 и воспользовался её при реализации задания. Создал модуль, обеспечивающий взаимодействие клиента и сервера, реализовал их соединение протокол, формат передачи данных. Успешно объединил все три лабораторные работы в общий работающий проект.