Университет ИТМО

Факультет ПИиКТ

Дисциплина

Низкоуровневое программирование

Лабораторная работа №3.

Вариант - XML

Выполнил:

Абульфатов Руслан Мехтиевич P33312

Преподаватель:

Кореньков Юрий Дмитриевич

г. Санкт-Петербург, 2023 год

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

2 модуля:

Client – 2 лаба
Server – 1 лаба

На клиенте происходит считывание команды, парсинг и упаковка в XML, затем отправка на сервер. Сервер после принятия запроса развертывает XML, определяется с типом запроса, достает из XML нужные параметры и выполняет соответствующую операцию, после чего посылает ответ.

Пример работы программы

Добавление элемента

```
add/0/[string=qwe][int=10][double=50.22][bool=1]
<?xml version="1.0"?>
<add><tuple id="0"><tuple string="qwe" operand_1="=" int="10" operand_2="=" double="50.22" operand_3="=" bool="1" operand_4="="/></tuple></add>
```

Удаление элемента по id

```
remove/1
<?xml version="1.0"?>
<remove><tuple id="1"/></remove>

deleted!
```

Поиск элемента по id

```
find/1
<?xml version="1.0"?>
<find><tuple id="1"/></find>


=============== TUPLE 1 ===============
parent_id - 0
name - bbb
age - 15
height - 51.462
male - 0
```

Поиск детей по родительскому id

```
find/0/*
<?xml version="1.0"?>
<find><tuple id="0"><tuple id="*" operand_1=""/></tuple></find>

=============== TUPLE 1 ===============
parent_id - 0
name - bbb
age - 15
height - 51.462
male - 0
=============== TUPLE 0 ===============
parent_id - 0
name - aaa
age - 12
height - 68.356
male - 0
```

Пример неправильного запроса

```
your request?

add/1/[name=qwe][age=10][height=50.22][healthy=1]]
incorrect
```

Аспекты реализации

Пример упаковки запроса в XML:
«Найти все (*) элементы, где родитель id - 0 (ну конкретно 0 значит, что его нет)»

```
find/0/*
<?xml version="1.0"?>
<find><tuple id="0"><tuple id="*" operand_1=""/></tuple></find>

=============== TUPLE 1 ===============
parent_id - 0
name - bbb
age - 15
height - 51.462
male - 0
=============== TUPLE 0 ===============
parent_id - 0
name - aaa
age - 12
height - 68.356
male - 0
```

Передача данных происходит через API ОС.

```c
#include "../include/inet.h"

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);
    }
}

```

```c
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 failed\n");
        exit( status: EXIT_FAILURE);
    }
    if (res == -1) {
        perror( s: "inet failed");
        exit( status: EXIT_FAILURE);
    }
}

```

Сам интерфейс взаимодействия с сервером

```c
int main(int argc, char **argv) {

    while (1) {

        int size_tree;
        xmlDocPtr request_tree;

        printf( format: "your request?\n\n");

        char req[MAX_REQUEST_SIZE];
        fgets( s: req,  n: MAX_REQUEST_SIZE,  stream: stdin);

        struct request *request = malloc( size: sizeof(struct request));
        enum parser_status status = parse_request(req, request);
        if (status != PARSE_OK) {
            printf( format: "parse error\n");
            exit(status);
        }

        request_tree = xmlNewDoc(BAD_CAST "1.0");
        status = get_xml(request, request_tree);
        xmlChar *str_tree = (xmlChar *) malloc(sizeof(xmlChar) * MAX_REQUEST_SIZE);
        xmlDocDumpMemory(request_tree, &str_tree, &size_tree);

        printf("%s\n", (char *) str_tree);

        if (status == WRAP_OK) sendRequest(atoi(argv[1]), size_tree, (char *) str_tree);

        free( ptr: str_tree);
        free_request(request);
```

Функции взаимодействия клиента с сервером

```
27
28  void get_request_view(struct request *request) {
29      int count_tab = 1;
30      struct attribute *attribute = request->attributes;
31
32      printf( format: "\naction - %s\n\n", request->operation);
33      printf( format: "id - %s\n", request->parent_id);
34
35      if (request->star) {
36          print_tab( count: &count_tab);
37          printf( format: "all nodes - %s\n", request->star);
38      }
39
40      if (attribute) print_attribute(attribute, &count_tab);
41  }
42
```

Вывод: Я успешно завершил интеграцию клиентской и серверной частей для взаимодействия по сети. В рамках этого проекта я освоил основы работы с библиотекой libxml для упаковки запросов в формат XML на клиентской стороне. Кроме того, я реализовал "usecase" на серверной стороне, таким образом обеспечивая обработку полученных запросов.

Эта лабораторная работа, позволила мне объединить знания низкоуровневого программирования с изучением сетевого взаимодействия на языке C через API ОС.