Университет ИТМО
Факультет ПИиКТ

Низкоуровневое программирование

Лабораторная работа №3

Работу выполнил:
Абузов Ярослав

Группа:
P33302

Вариант:
XML

Преподаватель:
Кореньков Ю. Д.

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. Протокол должен включать представление информации о командах создания, выборки, модификации и удаления данных в соответствии с данной формой, и результатах их выполнения.

Задачи:

1. Изучить выбранную библиотеку
2. На основе существующей библиотеки реализовать модуль, обеспечивающий взаимодействие
3. Реализовать серверную часть в виде консольного приложения
4. Реализовать клиентскую часть в виде консольного приложения

Детали реализации:

Для работы с XML исползовалась мощная библиотека libxml2. Она предоставляет множество функций и струкутур для работы с XML форматом. Взаимодействие с ней происходит очень просто (примеры ниже). Для создания TCP соединения использованы простые стандартные linux сокеты (там все стандартно, по первой ссылке в гугле).

Общие концепции лабораторной работы:

- Сервер слушает клиентов и работает с файлом базы данных ZGDB
- Клиент подключается к серверу (клиент может быть только один)
- Клиент может отправлять запросы ZPath, полученные от пользователя.
- После получения AST, клиент преобразует его в xml и отправляет на сервер в виде строки.
- Сервер получает xml строку и преобразует её в формат библиотеки для работы.
- Сервер определяет запрос и разбирает xml для получения необходимых аргументов.
- Сервер выполняет запрос с помощью функций ZGDB
- Результат пакуется в xml и отправляется клиенту в виде строки.
- Клиент преобразует строку в формат библиотеки libxml2 и выводит результат в читаемом виде.
- И все по новой

Схема отправки следующая:

- Преобразуем внктренний формат либы в char*
- Сначала отправляем пакет с размером этой строки
- Затем саму строку

И так в обе стороны – просто и быстро.

```
bytes_read = recv( fd: sock,  buf: &msgSize,  n: sizeof(int),  flags: 0);
if (bytes_read <= 0)
    break;

char buf[msgSize];
bytes_read = recv( fd: sock, buf,  n: msgSize,  flags: 0);
if (bytes_read <= 0)
    break;

xmlDocPtr request = xmlReadMemory( buffer: buf,  size: msgSize,  URL: 0,  encoding: NULL,  options: XML_PARSE_RECOVER);
xmlDocPtr answer = executeZgdbFromXml( doc: request,  file: pFile);
xmlFreeDoc( cur: request);

xmlChar* outXml;
int size;
xmlDocDumpMemory( cur: answer,  mem: &outXml, &size);
msgSize = size;

char* out = (char*) outXml;

send( fd: sock,  buf: &msgSize,  n: sizeof(int),  flags: 0);
send( fd: sock,  buf: out,  n: msgSize,  flags: 0);
```

Часть кода сервера для обработки запроса и отправки ответа

```
int sock, listener;
struct sockaddr_in addr;
size_t bytes_read;

listener = socket( domain: AF_INET,  type: SOCK_STREAM,  protocol: 0);
if (listener < 0) {
    perror( s: "socket");
    exit( status: 1);
}

addr.sin_family = AF_INET;
addr.sin_port = htons( hostshort: port);
addr.sin_addr.s_addr = htonl( hostlong: INADDR_ANY);
if (bind( fd: listener,  addr: (struct sockaddr*) &addr,  len: sizeof(addr)) < 0) {
    perror( s: "bind");
    exit( status: 2);
}

listen( fd: listener,  n: 1);
```

Создание сокета сервера

```
xmlDocPtr executeZgdbFromXml(xmlDocPtr doc, zgdbFile* file) {
    xmlNodePtr root = xmlDocGetRootElement(doc);
    xmlChar* string = xmlGetProp( node: root,  name: BAD_CAST "type");
    xmlDocPtr answer = xmlNewDoc( version: BAD_CAST "1.0");
    xmlNodePtr rootAnswer = xmlNewNode( ns: NULL,  name: BAD_CAST "answer");
    xmlDocSetRootElement( doc: answer,  root: rootAnswer);

    if (!xmlStrcmp( str1: string,  str2: BAD_CAST "add")) {
        createStatus status = executeAddFromXml(root, file);
        xmlNewProp( node: rootAnswer,  name: BAD_CAST "type",  value: BAD_CAST "createStatus");
        switch (status) {
            case CREATE_OK:
                xmlNewChild( parent: rootAnswer,  ns: NULL,  name: BAD_CAST "info",  content: BAD_CAST "CREATE OK");
                break;
            case OUT_OF_INDEX:
                xmlNewChild( parent: rootAnswer,  ns: NULL,  name: BAD_CAST "info",  content: BAD_CAST "OUT OF INDEX");
                break;
            case CREATE_FAILED:
                xmlNewChild( parent: rootAnswer,  ns: NULL,  name: BAD_CAST "info",  content: BAD_CAST "CREATE FAILED");
                break;
        }
```

Обработка запросов, точнее – делегация другим функциям (здесь ADD) и подготовка ответа

```
if (tree->type == AST_ADD) {
    xmlNewChild( parent: root,  ns: NULL,  name: BAD_CAST "document",  content: BAD_CAST tree->docName);
    xmlNodePtr schema = xmlNewChild( parent: root,  ns: NULL,  name: BAD_CAST "schema",  content: NULL);
    astAddSchema* temp = tree->first;
    size_t size = 0;
    while (temp != NULL) {
        xmlNodePtr schemaElement = xmlNewChild( parent: schema,  ns: NULL,  name: BAD_CAST "schemaElement",  content: NULL);
        char valueChar[50];
        switch (temp->type) {
            case SCHEMA_TYPE_INT: {
                sprintf( s: valueChar,  format: "%d", temp->integer);
                xmlNewProp( node: schemaElement,  name: BAD_CAST "type",  value: BAD_CAST "INT");
                xmlNewProp( node: schemaElement,  name: BAD_CAST "key",  value: BAD_CAST temp->name);
                xmlNewProp( node: schemaElement,  name: BAD_CAST "value",  value: BAD_CAST valueChar);
                break;
            }
            case SCHEMA_TYPE_DOUBLE: {
                sprintf( s: valueChar,  format: "%f", temp->dbl);
                xmlNewProp( node: schemaElement,  name: BAD_CAST "type",  value: BAD_CAST "DBL");
                xmlNewProp( node: schemaElement,  name: BAD_CAST "key",  value: BAD_CAST temp->name);
                xmlNewProp( node: schemaElement,  name: BAD_CAST "value",  value: BAD_CAST valueChar);
                break;
            }
            case SCHEMA_TYPE_BOOLEAN: {
                sprintf( s: valueChar,  format: "%d", temp->boolean);
                xmlNewProp( node: schemaElement,  name: BAD_CAST "type",  value: BAD_CAST "BOOL");
                xmlNewProp( node: schemaElement,  name: BAD_CAST "key",  value: BAD_CAST temp->name);
                xmlNewProp( node: schemaElement,  name: BAD_CAST "value",  value: BAD_CAST valueChar);
                break;
            }
            case SCHEMA_TYPE_STRING: {
                xmlNewProp( node: schemaElement,  name: BAD_CAST "type",  value: BAD_CAST "STR");
                xmlNewProp( node: schemaElement,  name: BAD_CAST "key",  value: BAD_CAST temp->name);
                xmlNewProp( node: schemaElement,  name: BAD_CAST "value",  value: BAD_CAST temp->string);
                break;
            }
        }
        temp = temp->next;
        size++;
    }
```


Часть кода клиента – упаковка схемы запроса ADD из AST (из 2 лабы) в XML

```
void createPathFromXml(xmlNodePtr pathNode, path* p) {
    xmlNodePtr firstStepElement = pathNode->last;
    while (firstStepElement->prev != NULL) {
        firstStepElement = firstStepElement->prev;
    }

    xmlNodePtr temp = firstStepElement;
    while (temp != NULL) {
        step s;
        xmlChar* pathTypeChar = xmlGetProp( node: temp,  name: BAD_CAST "pathType");
        xmlChar* stepTypeChar = xmlGetProp( node: temp,  name: BAD_CAST "stepType");
        xmlChar* nameChar = xmlNodeGetContent( cur: temp);
        if (nameChar == NULL) {
            nameChar = BAD_CAST "";
        }

        if (!xmlStrcmp( str1: pathTypeChar,  str2: BAD_CAST "abs")) {
            s.pType = ABSOLUTE_PATH;
        } else if (!xmlStrcmp( str1: pathTypeChar,  str2: BAD_CAST "rel")) {
            s.pType = RELATIVE_PATH;
        }

        if (!xmlStrcmp( str1: stepTypeChar,  str2: BAD_CAST "el")) {
            s.sType = ELEMENT_STEP;
        } else if (!xmlStrcmp( str1: stepTypeChar,  str2: BAD_CAST "doc")) {
            s.sType = DOCUMENT_STEP;
        }
        strcpy( dest: s.stepName,  src: (char*) nameChar);

        predicate pred;

        xmlNodePtr predicateNode = findNodeByName( rootNode: pathNode,  nodeName: BAD_CAST "predicate");
        xmlChar* sizeChar = xmlGetProp( node: predicateNode,  name: BAD_CAST "size");
        int64_t st = 0;
        str2long( out: &st,  s: (char*) sizeChar);
        if(st != 0) {
            predicate xml = createPredicateFromXml(predicateNode, &pred);
            s.pred = &xml;
        } else {
            s.pred = NULL;
        }
```

Сервер создают местную структуру пути (из 1 лабы) из полученного XML для выполнения какого-либо запроса

Примеры работы:

```
JOIN /"";
Request type: JOIN

    Path
    Step name: ; (absolute path, document step)

JOIN:
Document: test2
Document: test2
Document: test1
```

```
Client connected
<?xml version="1.0" encoding="UTF-8"?>
<request type="join">
  <predicate size="0"/>
  <path size="1">
    <step pathType="abs" stepType="doc">
      <predicate size="0"/>
    </step>
  </path>
</request>
```

Слева – клиент хочет посмотреть всех детей корня (сначала идет AST, затем полученный ответ в читаемом виде)
Справа – сервер получил запрос в виде xml строки и отправил ответ (вывод нужен для понимая того, что происходит)

```
ADD "zavar" (STR:@"name":"Yaroslav", INT:@"age":20) /"test2";
Request type: ADD
    New document name: zavar

    Schema
    STR:name:Yaroslav
    INT:age:20

    Path
    Step name: test2; (absolute path, document step)

CREATE:
CREATE OK
```

```
<?xml version="1.0" encoding="UTF-8"?>
<request type="add">
  <document>zavar</document>
  <schema size="2">
    <schemaElement type="STR" key="name" value="Yaroslav"/>
    <schemaElement type="INT" key="age" value="20"/>
  </schema>
  <path size="1">
    <step pathType="abs" stepType="doc">test2<predicate size="0"/></step>
  </path>
</request>
```

Слева – клиент хочет создать документ “zavar” у документа “test2”, получен ответ, что документ создан
Справа – полученный сервером запрос в виде xml

```
FIND /"test2"/"zavar";
Request type: FIND

    Path
    Step name: test2; (absolute path, document step)

        Step name: zavar; (absolute path, document step)

FIND:
Document: zavar
```

```
<?xml version="1.0" encoding="UTF-8"?>
<request type="find">
  <path size="2">
    <step pathType="abs" stepType="doc">test2<predicate size="0"/></step>
    <step pathType="abs" stepType="doc">zavar<predicate size="0"/></step>
  </path>
</request>
```

Проверяем, что документ «zavar» существует

```
UPDATE @"age" "lol" /"test2"/"zavar";
Request type: UPDATE
    Element name: age
    New value: lol

    Path
    Step name: test2; (absolute path, document step)

        Step name: zavar; (absolute path, document step)

UPDATE:
TYPE PARSE ERROR
```

```
<?xml version="1.0" encoding="UTF-8"?>
<request type="update">
  <element name="age" newValue="lol"/>
  <path size="2">
    <step pathType="abs" stepType="doc">test2<predicate size="0"/></step>
    <step pathType="abs" stepType="doc">zavar<predicate size="0"/></step>
  </path>
</request>
```

Попытка обновить элемент age в zavar. У него тип INT, поэтому строку «lol» в него не записать, о чем и говорит полученный ответ (ошибка).

```
FIND //@"key";
Request type: FIND

    Path
    Step name: key; (relative path, element step)

FIND:
Element: key, type: INT, value: 2, doc: test2
Element: key, type: INT, value: 2, doc: test2
Element: key, type: INT, value: 2, doc: test1
```

```
<?xml version="1.0" encoding="UTF-8"?>
<request type="find">
  <path size="1">
    <step pathType="rel" stepType="el">key<predicate size="0"/></step>
  </path>
</request>
```

Смотрим на все документы у корня с таким элементом

```
UPDATE @"key" "4" /"test2";
Request type: UPDATE
    Element name: key
    New value: 4

    Path
    Step name: test2; (absolute path, document step)

UPDATE:
UPDATE OK
```

```
<?xml version="1.0" encoding="UTF-8"?>
<request type="update">
  <element name="key" newValue="4"/>
  <path size="1">
    <step pathType="abs" stepType="doc">test2<predicate size="0"/></step>
  </path>
</request>
```

Обновим элемент у test2

```
FIND /@"key";
Request type: FIND

    Path
    Step name: key; (absolute path, element step)

FIND:
Element: key, type: INT, value: 4, doc: test2
Element: key, type: INT, value: 2, doc: test2
Element: key, type: INT, value: 2, doc: test1
```

Проверка обновления

```
DELETE /"test2"/"zavar";
Request type: DELETE

    Path
    Step name: test2; (absolute path, document step)

        Step name: zavar; (absolute path, document step)

DELETE:
DOCUMENT DELETED
```

```
<?xml version="1.0" encoding="UTF-8"?>
<request type="delete">
  <path size="2">
    <step pathType="abs" stepType="doc">test2<predicate size="0"/></step>
    <step pathType="abs" stepType="doc">zavar<predicate size="0"/></step>
  </path>
</request>
```

Удаляем zavar

Схема XML:

Каждый запрос содержится в теге <request> с пропсом type, который указывает на тип зпроса. У каждого запроса есть тег <path>, который содержит путь. В зависимости от запроса будт еще дополнительные теги:

- Add: <document> - имя создаваемого документа, <schema> - схема создаваемого документа (внутри теги <schemaElement> с типом, именем и значением элемента)
- Join - <predicate> - предикат для детей (внутри <predicateElement> с описанием предиката). Пример ниже:

```
<?xml version="1.0" encoding="UTF-8"?>
<request type="update">
  <element name="key" newValue="4"/>
  <path size="1">
    <step pathType="abs" stepType="doc">test2<predicate size="1"><predicateElement logOp="NONE"
inverted="0" type="BY_ELEMENT_VALUE" key="key" value="1"
compOperator="EQ"/></predicate></step>
  </path>
</request>
```

- Update - <element> - с информацией по обновляемому элементу
- Остальные запросы содержат только путь

Вывод:

В результате работы я объединил созданные до этого две систему в одну с помощью сети (оно даже работает). Научился использовать сокеты в C для работы с сетью и познакомился с библиотекой libxml2 для работы с XML в C.