Программа Сервер ввода-вывода

А.Н. Онин[1]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, alexii@niisi.ras.ru.

Аннотация. Статья содержит описание программы Сервер ввода-вывода (СВВ). Программа СВВ предназначена для буферизации и накопления в базе данных технологических параметров состояния. Программа СВВ используется как средство информационного взаимодействия ПЛК со SCADA-станцией в рамках функционирования АСУ ТП. Взаимодействие с программой СВВ происходит по протоколу REST-MLCP. Программа СВВ использует собственную СУБД для хранения данных в локальной файловой системе. Среда функционирования программы СВВ - операционная система семейства Linux.

Ключевые слова: протокол REST-MLCP, база данных, формат JSON, АСУ ТП

1. Введение

В статье рассматривается программа Сервер ввода-вывода (СВВ), приведены её описание, основные характеристики и преимущества перед аналогами.

Программа СВВ используется как средство информационного взаимодействия ПЛК со SCADA-станцией в рамках функционирования АСУ ТП.

Программа СВВ разрабатывалась как альтернативное решение программному-аппаратному комплексу «Сервер ввода-вывода» (ПАК СВВ) [1].

ПАК СВВ был построен на базе рабочей станции с микропроцессорной архитектурой Intel. Для запуска ПАК СВВ используется Docker-контейнер [4], под управлением которого запускаются СУБД «PostgreSQL», HTTP-сервер «Apache» и программа преобразования запросов протокола REST-MLCP в команды доступа к СУБД. Программные компоненты Docker-контейнер, СУБД «PostgreSQL», HTTP-сервер «Apache» являются свободно распространяемыми громоздкими продуктами, поддерживать которые крайне сложно. Связать компоненты ПАК СВВ между собой не является тривиальной задачей. Перед использованием ПАК СВВ необходимо заранее «вручную» создать все необходимые таблицы в БД. После создания таблиц их расширение дополнительными столбцами возможно только «вручную». Для функционирования ПАК СВВ используется немалый объём оперативной памяти (не менее 1 ГБ). Для программирования и настройки ПАК СВВ необходимо владеть следующим стеком технологий:

- знание языка программирования Perl;
- умение развёртывать и настраивать Docker-контейнер, СУБД «PostgreSQL», HTTP-сервер «Apache».

Программа СВВ лишена перечисленных выше недостатков, и имеет следующие преимущества:

- малый размер исполняемого образа программы (менее 1МБ);
- небольшой объём используемой оперативной памяти (от 2 до 5 МБ, зависит от интенсивности поступающих запросов);
- простота установки и запуска программы;
- автоматическое создание таблиц в БД на основе полученных запросов;
- автоматическое расширение таблиц дополнительными столбцами;
- быстрая обработка запросов (около 400 микросекунд затрачивается на выполнение одного запроса на запись в одну таблицу 100 столбцов);
- небольшой объём исходных текстов программы (около 100 КБ), написанных на языке программирования Си с использованием стандарта POSIX;
- возможность портирования программы под ОС РВ семейства Багет [5].

Программа СВВ использует свободно распространяемую библиотеку jansson для разбора и формирования текста в формате JSON. Исходные тексты библиотеки jansson включены в программу СВВ.

Недостатком программы СВВ является отсутствие доступа к БД при помощи стандартных SQL-запросов. Этот недостаток нивелируется созданием дополнительной БД под управлением свободно распространяемой СУБД, в которой будут зеркально отражаться данные, накопленные в БД программы СВВ.

2. Описание программы

Программа СВВ является серверным приложением, которое поддерживает соединения по сетевому протоколу HTTP и передачу данных по протоколу REST-MLCP [1].

Исполняемый модуль программы СВВ имеет название "SIO" и предназначен для функционирования под операционной системой семейства Linux.

Программа СВВ написана на языке программирования Си с использованием стандарта POSIX. Это позволяет легко портировать программу под ОС РВ семейства Багет.

Программа СВВ выполняет следующие действия:

- ожидает подключение клиента по сетевому протоколу HTTP;
- получает от клиента данные с помощью запроса *w_request* протокола REST-MLCP;
- сохраняет полученные данные в БД с помощью собственной СУБД;
- получает от клиента запрос данных по протоколу REST-MLCP;
- отправляет клиенту данные по протоколу REST-MLCP.

Программа СВВ поддерживает возможность параллельного обслуживания клиентских подключений.

Для каждого подключения создаётся отдельный поток управления.

Если клиент отправляет запрос на выполнение записи в БД, в этом случае этому потоку управления повышается приоритет. Это необходимо для того, чтобы запись в БД выполнялась в первую очередь, независимо от очереди запросов на чтение из БД.

Каждая операция записи в БД и чтения из БД является атомарной, т.е. текущая операция не будет прервана очередной операцией записи или чтения.

Структурная схема взаимодействия клиента с программой СВВ приведена на рисунке 1.

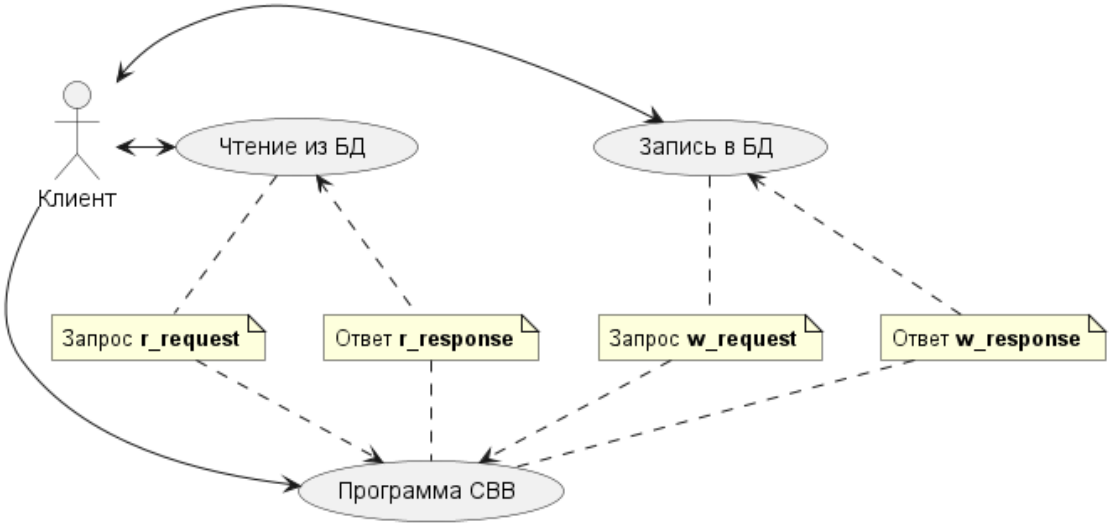

Рис. 1. Структурная схема взаимодействия клиента с программой СВВ

2.1. Обработка запросов по протоколу REST-MLCP

Программа СВВ поддерживает обработку следующих запросов, получаемых по протоколу REST-MLCP:

- *w_request* - запись данных в БД;
- *r_request* - чтение данных из БД.

Запрос "*w_request*" предназначен для записи одной строки в таблицу БД.

В запросе указывается номер таблицы БД и список тегов (столбцов таблицы) с их значениями.

Значение тега должно быть передано в формате double (64-разрядное вещественное число).

Каждая запись в таблицу помечается уникальной меткой времени.

В запросе необязательно указывать все столбцы таблицы, в которую выполняется запись.

Для неуказанных в запросе столбцов данные с текущей меткой времени будут отсутствовать.

Количество записей в таблице ограничено объёмом файловой системы, используемой для хранения данных в БД.

Запрос "w_request" передаётся программе СВВ в формате JSON и должен содержать следующие данные:

- номер таблицы БД (числовое поле "controller_id");
- список тегов с их значениями (поле "tags").

Каждый элемент списка тегов "tags" содержит два поля:

- название тега (символьное поле "tag");
- значение тега (числовое поле "value").

В качестве названия тега может использоваться любая последовательность символов из следующего набора: [A..Z], [a..z] [0..9], символ подчёркивания «_».

JSON-схема запроса "w_request" приведена на рисунке 2.

w_request			
controller_id	номер таблицы		
w_request	tags	tag	название тега
		value	значение тега

Рис. 2. JSON-схема запроса w_request

Запрос "w_request" обрабатывается в следующие порядке:

1) формируется метка времени записи "time";

2) в БД выбирается таблица, соответствующая номеру таблицы, указанному в запросе в поле "controller_id";

3) для каждого элемента поля "tags" в таблице выполняется поиск столбца, соответствующего названию тега (поле "tag");

4) в указанный столбец выполняется запись данных, соответствующих значению тега (поле "value");

5) текущей записи присваивается метка времени "time";

6) после выполнения записей всех тегов, указанных в запросе, формируется ответ "w_response" в формате JSON и отправляется клиенту по протоколу REST-MLCP.

Примечание.

1. Если в запросе указана таблица, которая не существует, тогда таблица создаётся автоматически.

2. Если в запросе указан тег, который отсутствует в таблице, тогда автоматически создаётся столбец одноимённый указанному тегу.

3. Метка времени - это 64-разрядное числовое значение времени в миллисекундах, отсчитанное от 01 января 1970, 0 часов, 0 минут, 0 секунд по Гринвичу.

4. Используется единая метка времени для пометки записей всех тегов, указанных в одном запросе.

Пример запроса "w_request" в формате JSON:

```
{
  "controller_id": 5,
  "w_request":
  {
    "tags":
    [
      {
        "tag": "tag1",
        "value": 3
      },
      {
        "tag": "tag2",
        "value": 4
      }
    ]
  }
}
```

В примере приведён запрос записи в таблицу № 5 двух тегов "tag1" и "tag2" со значениями 3 и 4, соответственно.

Ответ "*w_response*" предназначен для подтверждения выполнения запроса "w_request".

В ответе указывается статус ответа и текстовое сообщение, соответствующее статусу.

Ответ "w_response" передаётся клиенту в формате JSON и содержит следующие данные:

- статус (числовое поле "status");

- текстовое сообщение (символьное поле "message").

JSON-схема ответа "w_response" приведена на рисунке 3.

w_response	
status	признак выполнения запроса w_request
message	текстовое сообщение

Рис. 3. JSON-схема ответа w_response

В поле "status" передаётся признак обработки запроса. Поддерживаются следующие значения поля "status":

- число *0* - запрос успешно обработан;

- число *-1* - во время обработки запроса произошла ошибка.

В поле "message" передаётся текстовое описание признака обработки запроса.

Пример ответа "w_response", сформированный в результате успешной обработке запроса "w_request":

```
{
  "status": 0,
  "message": "tags push ok, database status true"
}
```

Пример ответа "w_response", сформированный в результате обнаружения ошибки во время обработки запроса "w_request":

```
{
  "status": -1,
  "message": "error"
}
```

Запрос "*r_request*" предназначен для чтения одного или нескольких столбцов из последней строки записи таблицы БД.

В запросе указывается номер таблицы БД и список тегов (столбцов таблицы).

В запросе необязательно указывать все столбцы таблицы, из которой должны быть прочитаны данные.

Значение для каждого прочитанного тега формируется в формате double (64-разрядное вещественное число).

Для каждого тега из таблицы БД извлекается последнее записанное значение.

Это означает, что извлечённые значения по набору тегов не обязаны быть с одинаковыми метками времени.

При этом метка времени в ответе не передаётся.

Для неуказанных в запросе столбцов данные из таблицы извлекаться не будут.

Для отсутствующих столбцов таблицы, в качестве значения тега, будет отправлено в ответе значение *null*.

Запрос "r_request" передаётся программе СВВ в формате JSON и должен содержать следующие данные:

- номер таблицы БД (числовое поле "controller_id");

- список тегов (поле "tags").

Каждый элемент списка тегов "tags" является символическим именем столбца таблицы, заключённым в двойные кавычки.

В качестве названия тега может быть использована любая последовательность символов из следующего набора: [A..Z], [a..z] [0..9], символ подчёркивания «_».

JSON-схема запроса "r_request" приведена на рисунке 4.

r_request		
controller_id	номер таблицы	
r_request	tags	массив названий тегов

Рис. 4. JSON-схема запроса r_request

Запрос "r_request" обрабатывается в следующие порядке:

1) в БД выбирается таблица, соответствующая номеру таблицы, указанному в запросе в поле "controller_id";

2) для каждого элемента поля "tags" в таблице выполняется поиск столбца, соответствующего названию тега;

3) из указанного столбца выполняется чтение последних записанных данных;

4) после чтения всех тегов, указанных в запросе, формируется ответ "r_response" в формате JSON и отправляется клиенту по протоколу REST-MLCP.

Примечание.

1. Если в запросе указана таблица, которая не существует, тогда таблица создаётся автоматически.

2. Если в запросе указан тег, который отсутствует в таблице, тогда для этого тега будет возвращено значение *null*.

Пример запроса "r_request" в формате JSON:

```
{
  "controller_id": 6,
  "r_request":
    {
      "tags": [ "tag1", "tag2" ]
    }
}
```

В примере приведён запрос чтения из таблицы № 6 двух тегов "tag1" и "tag2".

Ответ "r_response" предназначен для подтверждения выполнения запроса "r_request".

В ответе указывается статус ответа с текстовым сообщением, соответствующим статусу, и список тегов с их значениями.

Ответ "r_response" передаётся клиенту в формате JSON и содержит следующие данные:

- статус (числовое поле "status");

- текстовое сообщение (символьное поле "message");

- массив тегов (поле "tags").

JSON-схема ответа "r_response" приведена на рисунке 5.

r_response			
status	признак выполнения запроса r_request		
message	текстовое сообщение		
r_response	tags	tag	название тега
		value	значение тега

Рис. 5. JSON-схема ответа r_response

В поле "status" передаётся признак обработки запроса. Поддерживаются следующие значения поля "status":

- число *0* - запрос успешно обработан;

- число *-1* - во время обработки запроса произошла ошибка.

В поле "message" передаётся текстовое описание признака обработки запроса.

В поле "tags" передаётся массив тегом с их значениями.

Каждый элемент поля "tags" содержит следующие поля:

- название тега (столбца таблицы) (поле "tag");

- значение тега (поле "value").

Пример ответа "r_response", сформированный в результате успешной обработке запроса "r_request":

```
{
  "status": 0,
  "message": "tags are successfully fetched from database",
  "r_response":
  {
    "tags":
    [
      {
        "tag": "tag1",
        "value": 7
      },
      {
        "tag": "tag2",
        "value": 8
```

```
        }
      ]
    }
}
```

Пример ответа "r_response" сформированный в результате обнаружения ошибки во время обработки запроса "r_request":

```
{
  "status": -1,
  "message": "error"
}
```

2.2 Логическая структура БД программы СВВ

Логически структура БД состоит из таблиц, каждая таблица содержит один или более столбцов, каждый столбец содержит набор записей в виде блока данных с уникальной меткой времени.

Каждая таблица имеет уникальный номер и соответствующее этому номеру название.

Например, таблица с номером 1 будет иметь название "s0000000001_tags".

Номер таблицы используется в запросах "w_request", "r_request" в качестве значения поля "controller_id".

Номер таблицы может быть указан в диапазоне от *1* до *4 294 967 295*, включительно.

Максимальное допустимое количество таблиц - *100 000*.

Каждый столбец, в рамках одной таблицы, имеет уникальный номер и название.

Название столбца эквивалентно названию тега, указанного в запросах "w_request", "r_request".

Номер столбцу присваивается автоматически при создании столбца.

Максимальное допустимое количество столбцов в таблице – *16 777 214*.

Данные в столбец записываются при обработке запроса "w_request".

Формат данных - 64-разрядное вещественное число стандарта IEEE 754 [2].

Каждой записи данных присваивается уникальная метка времени - 64-разрядное целое число, указанное в миллисекундах, в соответствии со стандартом ISO 8601 [3].

Если в запросе "w_request" указаны несколько тегов, тогда будет использоваться единая метка времени для записей данных во все столбцы.

Логическая структура БД приведена на рисунке 6.

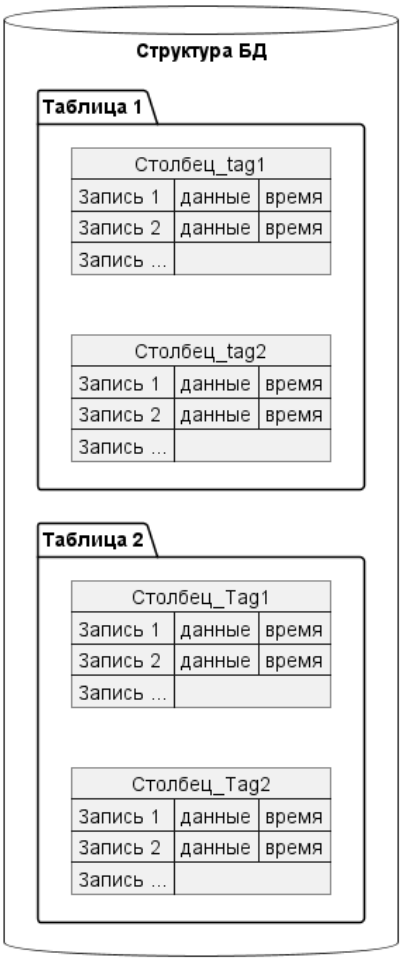

Рис. 6. Логическая структура БД

2.3 Файловая структура БД программы СВВ

Программа СВВ использует собственную БД, построенную на основе использования локальной файловой системы.

База данных формируется в файловой системе ОС Linux, в текущем каталоге запуска программы СВВ.

БД программы СВВ состоит из следующих компонент:

- каталоги БД;
- файлы в каталогах БД;
- файл конфигурации структуры БД.

Каждый каталог БД соответствует таблице БД. Название каталога эквивалентно названию таблицы.

Каталог БД создаётся при обработке запроса "w_request" в случае, если указанная в запросе таблица ещё не создана.

Файлы в каталоге БД создаются по мере добавления данных в столбцы соответствующей таблицы.

Файлы с данными имеют названия эквивалентные номеру дня, отсчитанного от 1 января

1970 года.

Например, при добавлении в таблицу данных 11 августа 2023 года в каталоге БД будет создан файл с названием "19580".

Программа СВВ выбирает данные из таблицы по метке времени.

Для ускорения поиска данных по метке времени создаётся индексный файл в каталоге БД.

Название индексного файла соответствует названию файла с данными, к имени которого добавляется суффикс ".i".

Программа СВВ во время функционирования использует файл конфигурации структуры БД.

Конфигурация структуры БД автоматически сохраняется в файле "*db-description.json*" в текущем каталоге запуска программы СВВ.

Пользователь может использовать свой файл конфигурации структуры БД, указав его при помощи ключа "-f" запуска программы СВВ.

2.4 Формат данных записей сохраняемых в файлах БД

Данные, полученные в запросах "w_request", сохраняются в файлах БД в формате, представленном на рисунке 7.

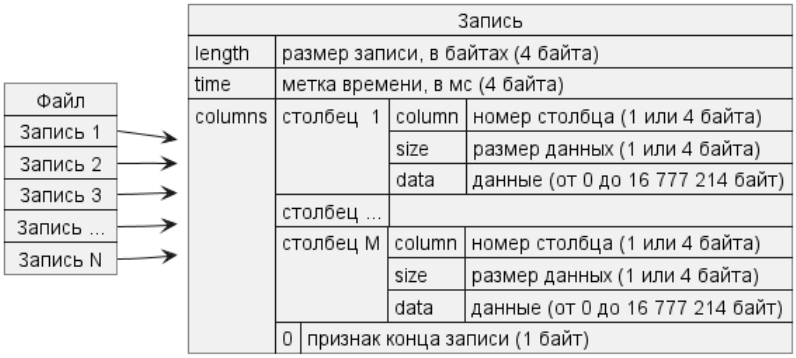

Рис. 7. Формат данных БД

Для каждой таблицы создаётся свой набор файлов. В одном файле сохраняется множество записей, полученных за сутки, соответствующих одной таблице.

Одной записи соответствует один запрос "w_request".

Поле записи "length" соответствует размеру записи без учёта размера поля "length".

Поле записи "time" формируется как уникальная временная метка, в миллисекундах, отсчитанная от начала суток по Гринвичу.

Поле записи "columns" является массивом столбцов с данными. Поле завершается признаком конца записи - один байт со значением "0".

Каждый элемент массива "columns" содержит данные одного столбца и состоит по трёх полей: "column", "size", "data".

Поле записи "column" содержит номером столбца.

Если номер столбца находится в диапазоне от 1 до 254, тогда поле "column" будет занимать 1 байт.

Если номер столбца более 254, тогда поле "column" будет занимать 4 байта. При этом первый байт поля будет равен 255, остальные 3 байта – содержат номер столбца.

Поле записи "size" содержит размер данных.

Если номер размер находится в диапазоне от 1 до 254 байт, тогда поле "size" будет занимать 1 байт.

Если номер размер более 254 байт, тогда поле "size" будет занимать 4 байта. При этом первый байт поля будет равен 255, остальные 3 байта – содержат размер данных.

Если данные целочисленные (или вещественные) и равны 0, тогда поле "size" будет равен "0", а поле "data" будет отсутствовать.

Если данные целочисленные (или вещественные) и равны 1, тогда поле "size" будет равен "255", а поле "data" будет отсутствовать.

Поле записи "data" содержит данные записи, соответствующего столбца.

Размер данных определён в поле "size".

Тип записи определён в поле "type" конфигурации структуры БД.

Все поля, в которых указано целочисленное значение, записываются в файле с порядком байт "LITTLE-ENDIAN".

2.5 Конфигурация структуры БД

Конфигурация структуры БД формируется в формате JSON и имеет структуру, приведённую на рисунке 8.

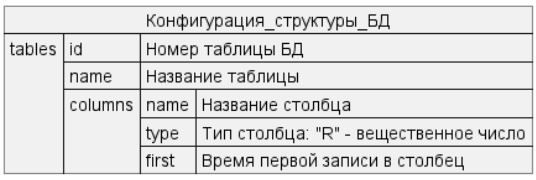

Рис. 8. JSON-схема конфигурации структуры БД

Конфигурация структуры БД состоит из массива таблиц "tables".

Каждая элемент таблицы состоит из следующих компонент:

- "id" - номер таблицы;
- "name" - название таблицы;
- "columns" - массив столбцов таблицы.

Каждый столбец таблицы состоит из следующих компонент:

- "name" - название столбца;
- "type" - тип столбца;
- "first" - время сохранения первой записи в столбец.

Программа СВВ поддерживает следующие типы столбцов:

- "R" - вещественное число (64 бита);
- "I" - целое число (до 64 бит);
- "T" - строка символов;
- "B" - данные типа BLOB.

Примечание. В запросе "w_request" могут передаваться данные только типа "R".

Пример конфигурации структуры БД приведён на рисунке 9.

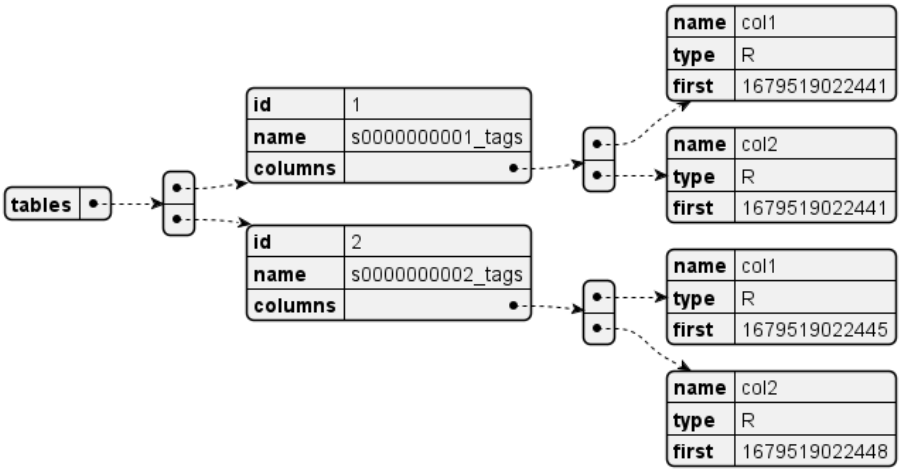

Рис. 9. Пример конфигурации структуры БД

Пример файла конфигурации структуры БД в формате JSON:

```
{
  "tables": [
    {
      "id": 1,
      "name": "s0000000001_tags",
      "columns": [
        {
          "name": "col1",
          "type": "R",
          "first": 1679519022441
        },
        {
          "name": "col2",
          "type": "R",
          "first": 1679519022441
        },
        {
          "name": "col3",
          "type": "R",
          "first": 1679519022441
        }
      ]
    },
    {
      "id": 2,
      "name": "s0000000002_tags",
      "columns": [
        {
          "name": "col1",
          "type": "R",
          "first": 1679519022445
        },
        {
          "name": "col2",
          "type": "R",
          "first": 1679519022448
        },
        {
          "name": "col3",
          "type": "R",
          "first": 1679519022448
        }
      ]
    }
  ]
}
```

2.6 Список тегов со значениями по умолчанию

Программа СВВ поддерживает привязку к имени тега значения по умолчанию. Привязка выполняется без указания названия таблицы. Другими словами – все привязки применимы ко всем таблицам.

Привязка срабатывает в случае чтения из таблицы, когда по указанному тегу (столбцу) отсутствуют данные. В этом случае, выполняется поиск значения тега по списку тегов, значения которых заданы по умолчанию. Найденное значение будет отправлено в ответе.

Список тегов со значениями по умолчанию должен быть сформирован в файле в формате JSON. JSON-схема списков тегов со значениями

по умолчанию приведена на рисунке 10.

Список_тегов_со_значениями_по_умолчанию		
defaults	tag	Название тега
	value	Значение тега по умолчанию

Рис. 10. JSON-схема списков тегов со значениями по умолчанию

Пример файла, содержащего список тегов со значениями по умолчанию:

```
{ "defaults":
 [
  { "tag": "col1", "value": 1.0 },
  { "tag": "col2", "value": 2.0 }
 ]
}
```

3. Заключение

Разработка программы СВВ имеет важное значение для организации легковесной СУБД в рамках обработки запросов по протоколу REST-MLCP.

В рамках работы была разработана новая структура организации БД. В результате в программе был реализован простой и быстрый способ записи в таблицы БД и реализован алгоритм быстрого поиска данных в таблицах.

Программа может быть доработана с целью расширения возможностей по обработке других типов запросов, получаемых в формате JSON (помимо поддержки протокола REST-MLCP).

Программа СВВ может быть легко портирована на микропроцессорную архитектуру MIPS под управлением ОС РВ семейства Багет.

В настоящий момент программа успешно используется как средство информационного взаимодействия ПЛК со SCADA-станцией для контроля технологического процесса объектов управления свалочных полигонов.

Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).

I/O Server Program

A. Onin

Abstract. The article contains a description of the I/O Server program (SIO). The SIO program is intended for buffering and accumulation of technological state parameters in the database. The SIO program is used as a means of information interaction between the PLC and the SCADA station used to control the technological process of the control object. Interaction with the SIO program occurs via the REST-MLCP protocol. To store data in the file system, the SIO program's own DB is used. The operating environment of the SIO program is an operating system of the Linux family.

Keywords: protocol REST-MLCP, database, format JSON

Литература

1. А. И. Грюнталь, К. Г. Нархов. Методы удаленной отладки ПЛК в среде ТСАГ СПО. Труды НИИСИ РАН, 2020, Т.10, № 5, 120–126.

2. Международный стандарт IEEE 754–1985 - IEEE Standard for Binary Floating-Point Arithmetic.

3. Международный стандарт ISO 8601:2004(E).

4. Э. Моуэт. Использование Docker. М, ДМК Пресс, 2017.

5. А. Н. Годунов, В. А Солдатов. Операционные системы семейства Багет (сходства, отличия и перспективы) «Программирование», Т 40, (2014) № 5, с. 68–76.