1. Введение

В ходе комплексного тестирования программных средств возникает необходимость интерпретации результатов множества тестов, написанных по различным правилам и для различных целей. Для решения задачи автоматизации запуска, сбора информации и интерпретации результатов тестирования необходимо привести интерфейсную часть всех тестирующих программ к единообразному виду позволяющему с наименьшими затратами решать поставленную задачу. Для данных целей предлагается использовать единую библиотеку с небольшим прикладным программным интерфейсом (API), исключающую возможность административного взаимонепонимания при реализации правил для обработки входных параметров и систематизации выходных данных в средствах тестирования и диагностики. При этом, в силу того что приложения тестирования могут взаимодействовать с оборудованием, необходимо обеспечить полную поддержку многопоточности в такой библиотеке.

2. Специальная часть

2.1. Анализ исходных требований для разрабатываемой библиотеки обработки входных параметров и систематизации выходных данных

Так как в исходных требованиях к разрабатываемой библиотеке указана необходимость совместимости с архитектурами SPARC V8, SPARC V9, i386, x86_64, то следует обеспечить независимость данного программного продукта от архитектуры процессора. Это достигается путем использования при разработке языка программирования высокого уровня (Си), обеспечивающего создание кросс-платформенного приложения.

Си является стандартизированным языком программирования. Это дает гарантию того, что однажды написанная, программа может быть использована на разных архитектурах. Ответственность за адаптацию

высокоуровневых конструкций языка программирования к особенностям конкретной архитектуры берет на себя компилятор с этого языка для данной конкретной архитектуры. В данной работе использовался компилятор GNU Compiler Collection (обычно используется сокращение gcc), поддерживающий большое количество архитектур, в том числе и требуемые. К тому же данный компилятор обеспечивает возможность кросс-компиляции, то есть создание исполняемого файла (в данном случае – библиотеки) для платформы отличной от той, на которой запускается компиляция.

Для осуществления кросс-компиляции в gcc обычно применяется команда *<архитектура>-gcc* (например: *SPARC-gcc*). Существует также и второй вариант: использование команды *gcc* с ключом *–b <архитектура>*.

Для каждой архитектуры в gcc имеется свой список опций. В частности для компиляции под процессоры архитектуры SPARC V8 необходимо указать ключ *–mcpu=v8*, а для 9 версии – ключ *–mcpu=v9*.

Основное различие 8 и 9 версий архитектуры SPARC заключается в том, что в 9 версии добавлена поддержка 64–битной адресации. Также все целочисленные регистры SPARC V8 были расширены из 32-битных в 64-битные. Кроме того появились новые инструкции для работы с 64-битными операндами.

2.2. Разработка соглашений о вызовах функций библиотеки

2.2.1. Разработка соглашений о вызовах функций обработки ошибок работы библиотеки

2.2.2. Разработка соглашения о вызове функции инициализации библиотеки

Функцией инициализации библиотеки является функция tioInit. До её вызова запрещается вызов любой другой функции библиотеки, за

исключением функции tioGetVersion. В задачи tioInit входит не только выделения памяти и задание начальных значений для переменных, массивов и структур, без которых невозможно использовать другие функции разрабатываемой библиотеки, но и производит разбор входных параметров для программы тестирования. Функция принимает как "длинные" так и "короткие" параметры. Все параметры, ключи которых содержат больше одного символа за исключением символа двоеточия на конце, являются длинными, все прочие называются короткими. Ключ из одного символа так же может быть длинным.

Прототип функции tioInit:

```
int tioInit   (   const char* version,
                  const char* help,
                  const tio_param _param[],
                  int argc,
                  char *argv[]
              )
```

Как видно из прототипа функция принимает 5 параметров:

1. version - версия теста, для которого инициализируется библиотека;

2. help - короткое описание назначения теста;

3. _param[] - список параметров принимаемых приложением и тех ключей для параметров, что используются в данном приложении. Признаком конца списка параметров является структура tio_param у которой все поля имеют значение NULL. Поля структуры tio_param приводятся ниже;

4. argc - количество аргументов командной строки;

5. argv[] - список аргументов командной строки;

tio_param представляет собой структуру вида:

```
typedef struct _tio_param
{
   char *key;
   char *name;
```

```
    char* description;
} tio_param;
```

Где key — ключ, используемый при вызове из командной строки, name - имя параметра, используемое при взаимодействии приложения с библиотекой, а description - короткое пояснение для каких целей используется параметр.

В качестве имени параметра разрешается использовать любую последовательность символов, состоящую из букв, цифр, символов подчеркивания и знака минус длиной до 126 символов.

В качестве ключа разрешено использовать последовательность символов, начинающуюся с буквы или с цифры. В теле последовательности могут содержаться буквы, цифры и знак минус. Так же строка не должна совпадать со словами «help», «version» и начинаться с «tio-». Символы минус в начале ключа и двоеточие в его конце несут служебную информацию и интерпретируются.

2.2.3. Разработка соглашений о вызовах функций получения входных параметров программ тестирования

Для того чтобы автоматизировать получение параметров командной строки предлагается использовать семейство функций tioGet* и tioGetDef*, где вместо знака «*» должна быть подставлена одна из следующих букв, означающих какого типа будет возвращаемое значение:

- L – long
- D – double
- C – char
- S – char* (string)

Коды ошибок в результате работы функций содержатся в таблице 2.1, приведенной ниже.

TENOPAR	Параметр не зарегистрирован при инициализации библиотеки
TEINCTYPE	Параметр не может быть приведен к запрошенному типу
TENOTSET	Параметр не передан при вызове приложения.
TENES	Размер буфера недостаточно велик для помещения параметра
TEFAILL	Отказ по непонятным причинам

Таблица 2.1

Функции tioGetS и tioGetDefS

```
int tioGetS   (   const char* name,
                  char* buff,
                  size_t buff_len
              )
```

Функция получения параметра командной строки в форме последовательности символов. name – указатель на имя параметра, значение которого необходимо получить. buff – указатель на адрес памяти, куда функция поместит значение искомого параметра в виде последовательности символов. buff_len – переменная, содержащая значение максимальной длины строки.

Возвращает 0 в случае успешного выполнения. В противном случае возвращаемое значение примет вид кода ошибки из таблицы 2.1. При возникновении любой из ошибок функция tioGetS заносит в buff нулевой символ.

```
int tioGetDefS   (   const char* name,
                     const char* default,
                     char* buff,
                     size_t buff_len
                 )
```

Функция получения параметра командной строки в форме последовательности символов. name – указатель на имя параметра, значение которого необходимо получить. default – значение параметра, связанного с именем name по умолчанию. buff – указатель на адрес памяти, куда функция

поместит значение искомого параметра в виде последовательности символов. buff_len – переменная, содержащая значение максимальной длины строки.

В случае если значение, связанное с именем name получить не удалось, то в буфер buff присваивается значение параметра default.

Возвращает 0 в случае успешного выполнения. В противном случае возвращаемое значение примет вид кода ошибки из таблицы 2.1. При возникновении любой из ошибок функция tioGetDefS заносит в buff нулевой символ.

Функции tioGetL и tioGetDefL

```
long tioGetL   (   const char* name   )
```

Функция возвращает значение параметра командной строки, связанного с именем name. Значение должно быть расположено в промежутке от минимально допустимого для типа long до предшествующего максимально допустимому значению для типа long (от LONG_MIN до LONG_MAX-1). В случае если такого параметра нет, или значения параметра не находятся в указанном промежутке, или не могут быть приведены к типу данных long, возвращается максимально допустимое значение для типа long. Код ошибки в этом случае может быть получен с помощью функции tioGetError().

Возможные ошибки: TENOTSET и TEINCTYPE.

```
long tioGetDefL   (   const char* name,
                      const long default
                  )
```

Функция возвращает значение параметра командной строки, связанного с именем name. Значение должно быть расположено в промежутке от минимально допустимого для типа long до предшествующего максимально допустимому значению для типа long (от LONG_MIN до LONG_MAX-1). В случае если такого параметра нет, или значения параметра не находятся в указанном промежутке, или не могут быть приведены к типу данных long, возвращается значение по умолчанию присвоенное при вызове функции параметру default. Код ошибки в этом случае может быть получен с помощью функции tioGetError().

Возможные ошибки: TENOTSET и TEINCTYPE.

Функции tioGetC И tioGetDefC

unsigned char tioGetC (const char* name)

Функция возвращает значение символа, переданного из командной строки и связанного с именем name. В случае если возвращаемое значение не может быть приведено к типу unsigned char, возвращаемое значение будет иметь вид максимально допустимого числа для этого типа данных.

Код ошибки может быть получен при помощи вызова tioGetError. Возможные ошибки: TENOTSET и TEINCTYPE.

```
unsigned char tioGetDefC ( const char* name,
                           const unsigned char default
                         )
```

В случае успешного завершения функции, возвращаемое значение будет равно значению переданному из командной строки и связанному с именем name. В случае, если получить значение, связанное с именем name не удалось, то возвращаемое значение будет взято из параметра default.

Код ошибки может быть получен при помощи вызова tioGetError. Возможные ошибки: TENOTSET и TEINCTYPE.

Функции tioGetD И tioGetDefD

double tioGetD (const char* name)

Функция возвращает число с плавающей запятой, переданное в программу с параметром name. Значение числа может быть любым допустимым для переменной в формате double, за исключением значения максимально допустимого для данного типа данных. В случае неуспешного выполнения, возвращаемое значение принимает вид максимально возможного значения для типа double.

Код ошибки может быть получен при помощи вызова tioGetError. Возможные ошибки: TENOTSET и TEINCTYPE.

```
double tioGetDefD ( const char* name,
                    const double default
                  )
```

Функция получения параметра, связанного с именем name в форме числа с плавающей точкой, со значением по умолчанию. Значение числа может быть любым допустимым для переменной в формате double, за исключением значения максимально допустимого для данного типа данных. В случае если по каким либо причинам получить значение параметра по его имени не удалось, функция возвращает значение по умолчанию, определенное в параметре default.

Код ошибки может быть получен при помощи вызова tioGetError. Возможные ошибки: TENOTSET и TEINCTYPE.

2.2.4. Разработка соглашений о вызовах функций обработки выходных данных программ тестирования

Функции вывода делятся на два типа: функции строчного вывода и функции табличного вывода.

Функции табличного вывода.

Для предоставления данных в табличной форме определено следующее семейство функций:

- void* tioTableBegin (const char* format, …);
- void* tioTableRecord (void *td, …);
- int tioTableEnd(void *td).

Первая функция предназначена для инициализации таблицы, а так же для задания количества столбцов и их заголовков. В том числе в функции tioTableBegin происходит определение для каждого столбца типа данных, которые он будет содержать в себе.

Параметр format содержит строку символов, которая содержит в себе список имен столбцов таблицы, разделенных знаком амперсанд (&). В случае если знак амперсанд является частью имени столбца, необходимо использовать последовательность символов, состоящих из двух амперсандов

подряд. Далее в прототипе функции идет переменный список параметров, количество параметров которого зависит от количества столбцов таблицы. Значения этих параметров определяют типы значений соответствующих столбцов. В случае успеха возвращаемое значение является указателем на таблицу.

Функция tioTableRecord предназначена для добавления новой строки в таблицу, передаваемую с параметром td. Далее идет переменный список параметров, в каждом из которых содержатся значение соответствующей ячейки таблицы. В случае успеха возвращаемым значение, также как и в предыдущей функции, является указатель на таблицу.

Функция tioTableEnd является функцией, которая выводит в виде таблицы сформированные данные, полученные от вызовов предыдущих функций семейства tioTable. В том случае, если какие либо значения не могут быть представлены в одной строке ячейки таблицы, то функция добавляет столько строк в таблицу, сколько нужно для полного представления данного значения.

Между вызовами функций tioTable* разрешен вызов любых других функций библиотеки.

Функции строчного вывода

- int tioPrint(const char* messge);
- int tioPrintF(const char* template, ...);

2.3. Реализация функций разрабатываемой библиотеки

2.4. Прототипирование среды исполнения подпрограмм библиотеки

Пример 1

Есть функция, решающая квадратное уравнение. Стоит задача протестировать правильно ли она вычисляет корни. Для тестирования возьмем уравнение вида $x^2 + 2x - 3 = 0$. Значит параметр a равен 1,

параметр b равен 2 и параметр c равен -3. Известно, что корнями данного уравнения являются 1 и -3. Для того чтобы что бы убедиться в корректности работы функции решения квадратных уравнений, напишем тест, использующий функции разработанной библиотеки.

Параметры a, b, c, первый корень, второй корень передаются при вызове теста из командной строки. Строка, запускающая тест, должна выглятеть так:

имя_пользователя@имя_компьютера:/путь_к_тесту$ ./quadratic-equation –a 1 –b 2 –c -3 --root1 1 --root2 -3

Программа теста считывает входные параметры, запускает тестируемую функцию с параметрами a, b, c. Получившиеся результаты работы фукнции решения квадратного уравнения сравнивает с параметрами root1 и root2. На основании этого тестирующая программа принимает решение тест пройден успешно, или тестируемая функция дает неверные результаты.

3. Технологическая часть

3.1. Профилирование разрабатываемого программного обеспечения

3.2. Анализ производительности библиотеки интерфейсов

3.3. Отладка и тестирование разрабатываемой библиотеки

4. Охрана труда и окружающей среды. Разработка мероприятий по обеспечению благоприятных санитарно-гигиенических условий труда инженера

5. Экономическая часть. Обоснование экономической эффективности разработки библиотеки функций унификации процессов обработки входных параметров и систематизации выходных данных

6. Заключение