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. Далее в записке к дипломной работе разрабатываемая библиотека будет называться библиотекой *libtio*.

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.

Аналогично для семейств архитектур *i386* и *x86-64* у *gcc* имеется свой набор опций.

Если заранее известно на процессоре какой архитектуры будет использоваться данная библиотека, то для оптимизации её работы можно при кросс-компиляции использовать ключ *–mtune=<cpu-type>*, где *<cpu-type>* - это тип конкретной архитектуры процессора.

Основной отличительной особенностью архитектуры x86-64 от i386 является поддержка 64-битных регистров общего назначения, 64-битных арифметических и логических операций над целыми числами и 64-битных виртуальных адресов.

Процессоры архитектуры *x86-64* поддерживают два режима работы: *Long mode* («длинный» режим) и *Legacy mode* («наследственный», режим совместимости с 32-битным *x86*), которые можно явно задать при компиляции, используя ключи *–m64* и *–m32* соответственно. Следовательно если нужно чтобы библиотека запускалась и на архитектуре *i386* и на архитектуре *x86-64* нужно использовать ключ *–m32*.

Важным отличием *SPARC* архитектур от архитектур *i386* и *x86-64* является порядок записи байт. *SPARC* использует *big-endian* (порядок байт от старшего к младшему), а *i386* и *x86_64* – *little-endian* (от младшего к старшему). Запись многобайтового числа из памяти компьютера в файл или передача по сети требует соблюдения соглашений о том, какой из байтов является старшим, а какой младшим, так как прямая запись ячеек памяти приводит к возможным проблемам при переносе приложения с платформы на платформу. Для разрабатываемой библиотеки был принят порядок байт от старшего к младшему (*big-endian*), так как он является стандартным для протокола *TCP*/*IP* (протокола управления передачей по сети).

В исходных требованиях указано, что разрабатываемая библиотека будет использоваться в операционных системах использующих стандарты *POSIX*.

POSIX (англ. *Portable Operating System Interface for Unix* — Переносимый интерфейс операционных систем *Unix*) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой. Стандарт создан для обеспечения совместимости различных *UNIX*-подобных операционных систем и переносимости прикладных программ на уровне исходного кода.

С точки зрения программиста-разработчика следование стандарту *POSIX* заключается в использовании заголовочных файлов и системных вызовов языка Си, которые должны быть предоставлены соответствующей стандарту системой.

Библиотека должна быть написана на языке высокого уровня «Си» в соответствии со спецификацией *C99*.

C99 — современный стандарт языка программирования Си. Определен в ISO/IEC 9899:1999. Является развитием стандарта *C89*.

В *C99* было добавлено несколько новых возможностей, а также удалены лишние.

Добавленные средства:

- Встраиваемые функции (объявленные с ключевым словом *inline*)
- Место, в котором возможно объявление переменных, больше не ограничено глобальной областью видимости и началом составного оператора (блока)
- Несколько новых типов данных, включая *long long int*, дополнительные расширенные целые типы, явный логический тип данных, а также комплексный тип (*complex*) для представления комплексных чисел
- Массивы переменной длины (*variable-length arrays*)
- Поддержка однострочных комментариев, начинающихся с //, как в *C++*
- Новые библиотечные функции, как, например, *snprintf*
- Новые заголовочные файлы, такие как *stdbool.h* и *inttypes.h*
- Типовые математические функции (*tgmath.h*)
- Улучшена поддержка стандарта IEEE 754-2008
- Составные константы
- Поддержка вариативных макросов (макросов переменной арности)
- Смягчение (*restrict*) ограничений для более агрессивной оптимизации кода

Некоторые удаленные средства:

Самым заметным "излишеством", удаленным при создании *C99*, было правило "неявного *int*". В *C89* во многих случаях, когда не было явного указания типа данных, подразумевался тип *int*. А в *C99* такое не допускается.

Также удалено неявное объявление функций. В *C89*, если функция перед использованием не объявлялась, то подразумевалось неявное объявление. А в *C99* такое не поддерживается. Если программа должна быть совместима с *C99*, то из-за двух этих изменений, возможно, придется немного подправить код.

C99 является большей частью обратно совместимым с *C90*, но вместе с тем в некоторых случаях является более жёстким. В частности, объявление без указания типа больше не подразумевает неявное задание типа *int*. Комитет по стандартизации языка Си решил, что для компиляторов будет более важным определять пропуск по невнимательности указания типа, чем «тихая» обработка старого кода, полагавшаяся на неявное указание *int*.

GCC и другие компиляторы языка Си поддерживают многие нововведения стандарта *C99*. Тем не менее, ощущается недостаточная поддержка стандарта со стороны крупных производителей средств разработки, таких как *Microsoft* и *Borland*, которые сосредоточились, в основном, на языке *C++*, так как *C++* обеспечивает функциональность, схожую с предоставляемой нововведениями стандарта.

Согласно *Sun Microsystems*, *Sun Studio* полностью поддерживает стандарт *C99*.

Интерпретатор языка Си *Ch* поддерживает основные особенности *C99* и свободно доступен в версиях для *Windows*, *Linux*, *Mac OS X Solaris*, *QNX* и *FreeBSD*.

Другие компиляторы с полной или частичной поддержкой стандарта *C99*

Digital Mars	*Intel C Compiler*	*Oracle Solaris Studio*
VPF	*LCC*	*Pelles C*
Open Watcom C		

2.2. Разработка соглашений о вызовах функций библиотеки

2.2.1. Разработка соглашений о вызовах функций обработки ошибок работы библиотеки

Все функции библиотеки, которые предназначенные для пользователей программистов и являющиеся экспортируемыми должны возвращать значение.

Функции, возвращающие указатель на некий тип данных, в случае ошибки должны возвращать нулевой указатель (NULL).

Функции, возвращающие числовое значение некого типа данных, при аварийном завершении возвращают максимально допустимое значение своего возвращаемого типа. Функции, возвращающий параметр которых имеет символьный тип, также относятся к возвращающим числовое значение. Стоит отметить, что в таком случае возвращаемый параметр является беззнаковым.

Код ошибки для последней вызванной функции библиотеки можно получить используя функцию *tioGetError*(), возвращаемым значением которой и будет код ошибки.

При возникновении ситуации из-за которой не может продолжаться нормальная работа функций библиотеки необходимо вызвать функцию

```
int tioDie   (   int status,
                 const char* buff,
             )
```

Вследствие её работы ресурсы памяти, занятые библиотекой будут освобождены.

Аргументы:

status - статус завершения приложения (TOFAIL, TOTESTNOTSTART)
msg - сообщение размещаемое в потоке ошибок

Сигнал TOFAIL означает, что программа тестирования завершилась с неудовлетворительным результатом.

Если приложение завершается со статусом TOTESTNOTSTART, то ошибка произошла ещё на стадии инициализации 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 _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 **Ошибка! Недопустимый**

объект гиперссылки. 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;

Ошибка! Недопустимый объект гиперссылки. представляет собой структуру вида:

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

} **Ошибка! Недопустимый объект гиперссылки.**;

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*».

Если при запуске программы тестирования используется ключ --*help* , то вместо выполнения теста в стандартный поток вывода будет представлена информация о списке ключей, доступных при вызове.

Если при запуске использовать ключ --*version*, то в стандартный поток вывода будет представлена информация о версии теста.

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 приведены ниже (Таблица 2.1).

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

Базовые возможности библиотеки рассмотрим на примере программы, которая тестирует функцию подсчета корней квадратного уравнения.

Есть функция, решающая квадратное уравнение.

Задача: протестировать являются ли корни, полученные на выходе функции, верными для уравнения заданного вида.

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 эталонный корень и второй эталонный корень передаются при вызове теста как параметры командной строки. Согласно данному уравнению, строка, запускающая тест, должна выглядеть так:

./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 функции решения квадратного уравнения выводит на экран вместе с эталонными значениями *root1* и *root2*.

Выполнение теста показано на рис. 2.1.

```
nwcfang@nf-lrti:~/current-task/prototypes/quadratic-equation$ ./quadratic-equation -a 1 -b 2 -c -3 --root1=1 --root2=-3
Тест написал: Гусев Михаил
Короткое описание теста:
Тестирование функции решения квадратного уровнения.
[RUN]: Запуск ./quadratic-equation
+-------------------------------------------------------------+-------------------------------------------------------------+
|Name                                                         |Value                                                        |
+-------------------------------------------------------------+-------------------------------------------------------------+
|Argument A                                                   |1                                                            |
+-------------------------------------------------------------+-------------------------------------------------------------+
|Argument B                                                   |2                                                            |
+-------------------------------------------------------------+-------------------------------------------------------------+
|Argument C                                                   |-3                                                           |
+-------------------------------------------------------------+-------------------------------------------------------------+
Сравнение эталонных и возвращаемых функцией корней
+-------------------------------------------------------------+-------------------------------------------------------------+
|Roots etalon                                                 |Roots from function                                          |
+-------------------------------------------------------------+-------------------------------------------------------------+
|1                                                            |1                                                            |
+-------------------------------------------------------------+-------------------------------------------------------------+
|-3                                                           |-3                                                           |
+-------------------------------------------------------------+-------------------------------------------------------------+
```

Рис. 2.1

Для разбора параметров командной строки, а также инициализации разработанной библиотеки использовалась функция *tioInit*. После вызова этой функции можно использовать функции библиотеки семейства *tioGet* для доступа к интересующим нас параметрам командной строки.

Наглядное представление выходных данных обеспечивается функциями семейства *tioTable*, позволяющих рисовать динамическую таблицу, в которой можно изменять заголовки и количество колонок, а также количество строк и тип данных в каждой ячейке строки.

С помощью ключа *--help*, переданному при вызове тестирующей программы, использующей библиотеку *libtio*, вместо выполнения теста на экран выведет список аргументов которые можно передать из командной строки (рис. 2.2).

```
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ ./quadratic-equation --help
Тест написал: Гусев Михаил
Короткое описание теста:
Тестирование функции решения квадратного уровнения.
Использование: ./quadratic-equation [КЛЮЧ]... [ФАЙЛ]...
This is test
 -a <ПАРАМЕТР>                Параметр A
 -b <ПАРАМЕТР>                Параметр B
 -c <ПАРАМЕТР>                Параметр C
 --root1 <ПАРАМЕТР>           Первый корень
 --root2 <ПАРАМЕТР>           Второй корень
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$
```

Рис. 2.2

При использовании ключа *--version* после команды, запускающей исполняемый файл программы тестирования, будет выведена справка о версии запускаемого теста (рис. 2.3).

```
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ ./quadratic-equation --version
Программа ./quadratic-equation версия v0.9 alpha
```

Рис. 2.3

Теперь рассмотрим программу тестирующую работоспособность COM-порта.

Программа может работать в трех режимах:

- режим «Клиент»;
- режим «Сервер»;
- режим «Клиент/Сервер».

Если выбран режим «Клиент», то программа работает по следующему алгоритму:

В течении двадцати секунд ожидает сообщение от программы «Сервер» о готовности к передаче данных. Если по истечению данного периода сообщение не получено, то тест завершается провалом. В случае если сообщение о готовности «Сервера» пришло, отправляется сообщение о готовности принимать данные. После чего принимаем пакеты.

Если выбран режим «Сервер», то программа работает так:

Вначале отправляется сообщение, что «Сервер» готов к передаче данных. Получив, ответ от «Клиента», что он готов к передаче, «Сервер» начинает передавать пакеты.

Режим «Клиент/Сервер» отличается от предыдущих тем, что создается д процесс потомок, который берет на себя роль «Сервера», а родитель будет работать как «Клиент».

В качестве входных параметров для тестирующей программы принимаются следующие ключи:

- «*-D*» – ключ имеет числовое значение. Продолжительность передачи пакетов;
- « *-m*» - ключ имеет числовое значение. Скорость передачи пакетов;
- «*-s*» - ключ имеет числовое значение. Размер передаваемого пакета;
- «*-d*» - программа будет работать в режиме Сервера, то есть отправлять пакеты Клиенту;
- «*-l*» - программа будет работать в режиме Клиента, то есть принимать пакеты от Сервера;
- «*-L*» - программа работает в режиме Клиент/Сервер, то есть пакеты будут отправляться и приниматься на одной и той же ЭВМ.

Эта программа была написана без использования библиотеки *libtio*. Метод обработки входных параметров описывался в отдельном файле и выглядел так:

```
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <unistd.h>

#include "config.h"

Configuration config = {
    115200, // speed
    -1,     // device fd
    "/dev/ttyUSB0", // device path
    1000,       // minimu m transfered data count
    CLIENTMODE, // mode of test
    0, // duration (nowtime is unused)
    1 // work mode
};

static int
```

```
calculate_configuration(Configuration *cfg)
{
    if (!cfg)
        return EINVAL;

    if (cfg->duration)
    {
        cfg->sendPacksLength = (cfg->duration * cfg-
>portSpeed / 8);
        cfg->duration = 0;
    }
    return 0;

}


int
write_configuration(Configuration *cfg, char **argv, int
argc)
{
    int opt;
    int already_typed = 0;

    if (!cfg || !argv ||  argc < 1)
        return EINVAL;

    while (-1 != (opt = getopt(argc, argv, "D:m:s:dlLh")))
    {
        switch(opt)
        {
        case 'D':
            cfg->duration = atol(optarg);
            if (cfg->duration <= 0)
                return EAGAIN;
            break;
        case 'm':
            cfg->portSpeed = atol(optarg);
            if (cfg->portSpeed <= 0)
                return EAGAIN;
            break;
        case 's':
            cfg->sendPacksLength = atol(optarg);
            if (cfg->sendPacksLength <= 0)
                return EAGAIN;
```

```
            break;
        case 'd':
            if (already_typed)
                return EAGAIN;
            cfg->serverClientMode = SERVERMODE;
            already_typed = 1;
            break;
        case 'l':
            if (already_typed)
                return EAGAIN;
            cfg->serverClientMode = CLIENTMODE;
            already_typed = 1;
            break;
        case 'L':
            if (already_typed)
                return EAGAIN;
            cfg->serverClientMode = CLIENTSERVERMODE;
            already_typed = 1;
            break;
        default:
            return EAGAIN;
        }
    }
    calculate_configuration(cfg);

    if (optind < argc)
        strcpy(config.DeviceName, argv[optind]);

    return 0;
}
```

Использование библиотеки libtio, а в частности функции *tioInit* позволяет сократить данный файл до вида:

```
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <unistd.h>

#include <tio.h>

#include "config.h"
```

```
Configuration config = {
    -1,         //device fd
    1000,       //minimu m transfered data count
    1           //work mode
};
static int
calculate_configuration(Configuration *cfg)
{
    if (!cfg)
        return EINVAL;

    if( tioGetDefL( "DURATION", 0 ) )
        cfg->sendPacksLength = ( tioGetL( "DURATION" ),
tioGetDefL( "PORTSPEED", 115200 ) / 8);
    return 0;
}
int
write_configuration(Configuration *cfg )
{
    cfg->sendPacksLength = tioGetDefL( "SENDPACKSLENGTH",
1000 );
    calculate_configuration(cfg);
    return 0;
}
```

Причем, чем больше ассортимент параметров командной строки, тем больше преимущество по времени у программиста, использующего библиотеку *libtio* пред программистом, пишущим код разбора входных параметров самостоятельно.

Стандартный поток вывода после выполнения тестирующей программы в режиме «Клиент/Сервер» показан на рис. 2.4

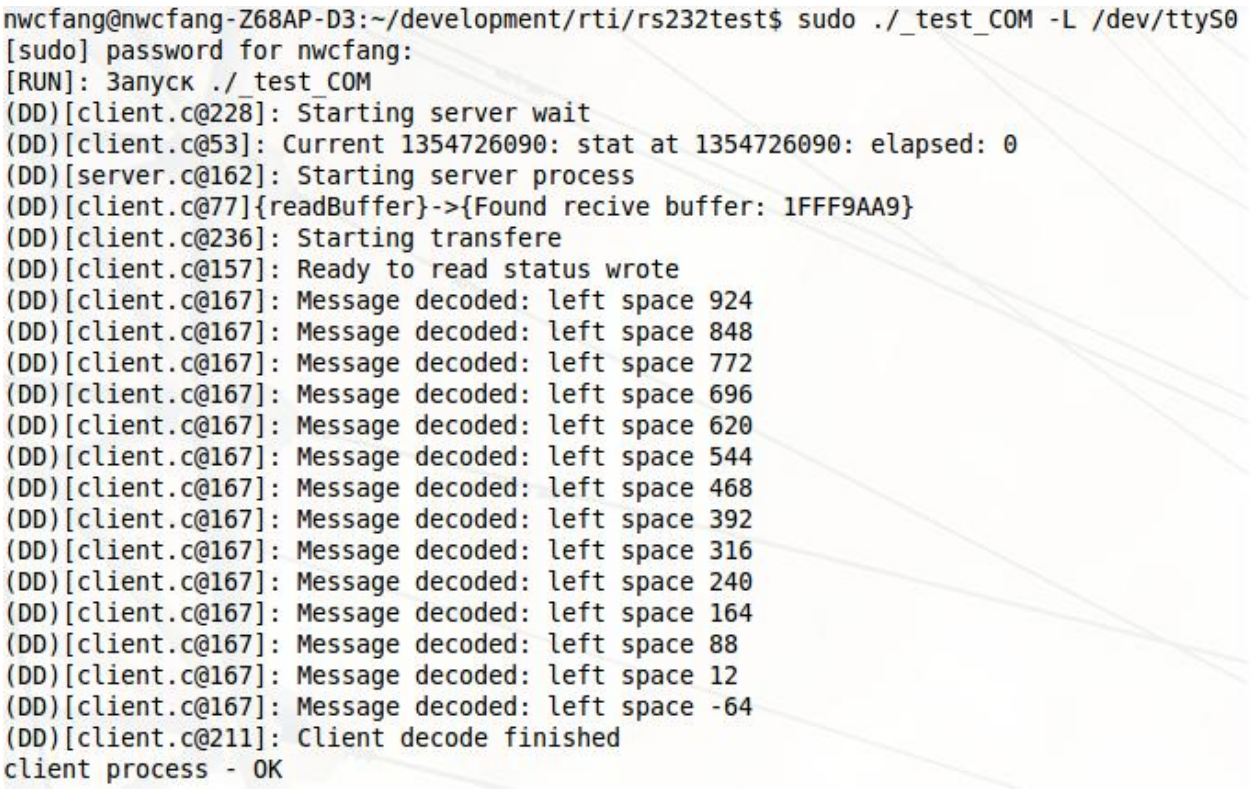

Рис. 2.4

3. Технологическая часть

3.1. Профилирование разрабатываемого программного обеспечения

3.2. Анализ производительности библиотеки интерфейсов

3.3. Отладка и тестирование разрабатываемой библиотеки

Отладка разработанной библиотеки производилась с помощью программы *GDB* (*GNU Debugger*), первоначально написанной Ричардом Столлмэном в 1988 году и являющейся свободным программным обеспечением.

GDB работает на многих *UNIX*-подобных системах и умеет производить отладку многих языков программирования, включая Си, *C++*, *Free Pascal*, *FreeBASIC*, *Ada* и Фортран.

Отладчик имеет средства для слежения и контроля за выполнением компьютерных программ. Пользователь может изменять внутренние

переменные программ и вызывать функции независимо от обычного поведения программы.

С версии 7.0 добавлена поддержка «обратимой отладки», позволяющей отмотать назад процесс выполнения, чтобы посмотреть, что произошло.

При разработке библиотеки *libtio* после добавления новой функции, проводилось автоматическое модульное тестирование по принципу черного ящика, что позволяло быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчало обнаружение и устранение таких ошибок.

Поток вывода после запуска модульных тестов изображен на рисунке

```
bin/runtests.sh bin
I have color
(TS): Run test: bin/test_all_out
[RUN]: Запуск ./bin/test_all_out
(II): Hello my darling
(II):This programm is 1-st test for me
(WW): I think It's rather prety
(WW):I programm it well be usefull
(EE): But i think it weel be great
(EE):It's over 2328
[PASS]: ./bin/test_all_out : Тест пройден успешно
(TS): Test  bin/test_all_out [PASS]
(TS): Run test: bin/test_basetioGetC
[RUN]: Запуск test_basetioGetC.c
[PASS]: test_basetioGetC.c : Тест пройден успешно
(TS): Test  bin/test_basetioGetC [PASS]
(TS): Run test: bin/test_basetioGetD
[RUN]: Запуск test_basetioGetD.c
[PASS]: test_basetioGetD.c : Тест пройден успешно
(TS): Test  bin/test_basetioGetD [PASS]
(TS): Run test: bin/test_basetioGetDefC
[RUN]: Запуск test_basetioGetDefC.c
[PASS]: test_basetioGetDefC.c : Тест пройден успешно
(TS): Test  bin/test_basetioGetDefC [PASS]
(TS): Run test: bin/test_basetioGetDefD
[RUN]: Запуск test_basetioGetDefD.c
[PASS]: test_basetioGetDefD.c : Тест пройден успешно
(TS): Test  bin/test_basetioGetDefD [PASS]
(TS): Run test: bin/test_basetioGetDefL
[RUN]: Запуск test_basetioGetDefL.c
[PASS]: test_basetioGetDefL.c : Тест пройден успешно
(TS): Test  bin/test_basetioGetDefL [PASS]
(TS): Run test: bin/test_basetioGetDefS
[RUN]: Запуск self
TRUE
[PASS]: self : Тест пройден успешно
(TS): Test  bin/test_basetioGetDefS [PASS]
(TS): Run test: bin/test_basetioGetL
[RUN]: Запуск self
[PASS]: self : Тест пройден успешно
(TS): Test  bin/test_basetioGetL [PASS]
(TS): Run test: bin/test_basetioGetS
[RUN]: Запуск self
TRUE
[PASS]: self : Тест пройден успешно
(TS): Test  bin/test_basetioGetS [PASS]
(TS): Run test: bin/test_error
Attempt to set error befor error initialization or after error free
(TS): Test  bin/test_error [PASS]
```

```
(TS): Run test: bin/test_error2
(TS): Test  bin/test_error2 [PASS]
(TS): Run test: bin/test_ErrorF
(EE):char = X long = 123 le = 1.154340e+01  float = 11.543400. 100 in oct = 144. This is string! and h = 7b, H = 7B. Happy% end!(TS): Test  bin/test_ErrorF [PASS]
(TS): Run test: bin/test_help
Использование: Test [КЛЮЧ]... [ФАЙЛ]...
Проверка описания программы.
 -k, -d, --list, --ls, --lst    List information about the FILEs (the current dire
                                ctory by default).  Sort entries alphabetically if
                                 none of -cftuvSUX nor --sort. Mandatory arguments
                                 to long options are mandatory for short options t
                                oo.
 --ip, --adress <ПАРАМЕТР>      show / manipulate routing, devices, policy routing
                                 and tunnels
 -s, -S, --sort <ПАРАМЕТР>      sort lines of text files
 --file, --fl <ПАРАМЕТР>        determine file type
 -t <ПАРАМЕТР>                  table view
(TS): Test  bin/test_help [PASS]
(TS): Run test: bin/test_longto
(TS): Test  bin/test_longto [PASS]
(TS): Run test: bin/test_output
(EE):267676836876843768763487686387683726863876876387638746873648763847687643 87jddhkjhfkjdhkjherw;jhl;kjelskjlkjelkjlkjefl;sj;lkej;lkj;lej;
ojlkhjlkewyp9ub oiuro;icnj8p2[oicnpoi2poi2p3ipoi2309878yrhjewhiouyfipdhlu3rpo8u7093kpojkfd09iupjrepwu09ufeoij09fueoijoifeu980j32oiy87y9jddh
lkjlkjesflkjlke
(EE):Символ = Y число 128 с плавающей: 1.154340e+01, ещё раз: 11.543400, 100 в окт: 144, строка: 8048700 в хексе: 7b, в ХЕКСЕ: 7B %THe end.
(EE):Символ = Z число 129 с плавающей: 1.154340e+01, ещё раз: 11.543400, 100 в окт: 144, строка: 8048700 в хексе: 7b, в ХЕКСЕ: 7B %THe end.
(TS): Test  bin/test_output [PASS]
(TS): Run test: bin/test_strcmp
(TS): Test  bin/test_strcmp [PASS]
```

```
(TS): Run test: bin/test_table
Cap string
Call "tioTableBegin".

Call "tioTableRecord".

Call "tioTableEnd".
+----------------------------+----------------------------+----------------------------+----------------------------+
|char                        |st&ring                     |double                      |string                      |
+----------------------------+----------------------------+----------------------------+----------------------------+
|r                           |An advantage of COM+        |23.70                       |Animated by Ryan Woodward   |
+----------------------------+----------------------------+----------------------------+----------------------------+
|e                           |An advantage of COM+        |43.90                       |The essence of COM is a langu|
|                            |                            |                            |age-neutral way of implementi|
|                            |                            |                            |ng objects that can be used i|
|                            |                            |                            |n environments              |
+----------------------------+----------------------------+----------------------------+----------------------------+
(TS): Test  bin/test_table [PASS]
(TS): Run test: bin/test_tioInit
Test start
[RUN]: Запуск self
[PASS]: self : Тест пройден успешно
(TS): Test  bin/test_tioInit [PASS]
```

```
(TS): Run test: bin/test_true
This test is allwayse good:)
(TS): Test  bin/test_true [PASS]
(TS): Run test: bin/test_utf
Амбивалентный
мбивалентный
бивалентный
ивалентный
валентный
алентный
лентный
ентный
нтный
тный
ный
ый
й
(TS): Test  bin/test_utf [PASS]
(TS): Run test: bin/test_version
Version 0.4.8, revision 209
(TS): Test  bin/test_version [PASS]
(TS): Run test: bin/fail_test2
test2
(TS): Test  bin/fail_test2 [PASS]
(TS): Run test: bin/fail_true
(TS): Test  bin/fail_true [PASS]
(TS): Run test: bin/fail_true2
(TS): Test  bin/fail_true2 [PASS]
(TS): All tests PASS
```

Сценарий командной оболочки, позволяющий автоматизировать процесс запуска тестов, представлен в приложении

Вначале сценария приписываем в переменную окружения LD_LIBRERY_PATH папку *./lib*. Это делается для того, чтобы модульные тесты искали скомпилированную библиотеку *libtio* в директории lib, находящуюся в корневой директории проекта. Далее проводим настройку цветов некоторых элементов вывода, таких как *TS, Test, Run test, PASS, FAIL* и т. д. Далее все исполняемые файлы, начинающиеся с *test_*, поочередно запускаются и если завершаются без ошибок, то в поток вывода печатается сообщение о том, что тест пройден. В противном случае – печатается сообщение о том, что тест провален.

Так же среди прочих тестов, имеются тесты, успешным выполнением которых является их завершение с определенным кодом ошибки. Название

таких тестов начинается с *fail_*. Такие тесты считаются успешно завершенными, если они возвращают значение равное значению из одноименного файла с типом *.result.*

В конце сценария переменная LD_LIBRERY_PATH возвращается в первоначальное значение.

Если все модульные тесты завершились успехом, то выводится сообщение, символизирующее отсутствие ошибок при автоматическом тестировании. Если хотя бы один тест провалился, то по сценарию выводится сообщение, что модульное тестирование не прошло успешно.

В ходе функционального тестирования было выявлено, что функция *tioTableEnd* отображает таблицу некорректно, если в полях таблицы используются символы кириллицы (рис. 3.1).

```
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ ./quadratic-equation -a 1 -b 2 -c -3 --root1=1 --root2=-3
[RUN]: Запуск ./quadratic-equation
+-------------------------------------------------+-------------------------------------------------+
|Имя параметра                                    |Значение                                         |
+-------------------------------------------------+-------------------------------------------------+
|Аргумент A                                       |1                                                |
+-------------------------------------------------+-------------------------------------------------+
|Аргумент B                                       |2                                                |
+-------------------------------------------------+-------------------------------------------------+
|Аргумент C                                       |-3                                               |
+-------------------------------------------------+-------------------------------------------------+
Сравнение эталонных и возвращаемых функцией корней
+-------------------------------------------------+-------------------------------------------------+
|Эталонные корни                                  |Корни, посчинанные функцией                      |
+-------------------------------------------------+-------------------------------------------------+
|1                                                |1                                                |
+-------------------------------------------------+-------------------------------------------------+
|-3                                               |-3                                               |
+-------------------------------------------------+-------------------------------------------------+
[PASS]: ./quadratic-equation : Тест пройден успешно
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ gvim quadratic-equation
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ gvim source.c
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$
```

Рис. 3.1

При отладке было установлено что, так как при работе с символьными данными (строками) по умолчанию используется кодировка *UTF-8*, а это значит, что для представления латинских символов требуется 1 байт, тогда как для представления символов кириллицы необходимо отводить под каждую букву 2 байта. В *tioTableEnd* это не было учтено.

```
@@ -345,7 +345,7 @@ int tabRow( void **strings, int
*bufType, int countColum, int lenColCon )

     /*Calculation number of extra lines of the array*/
     for( i = 0; i < countColum; ++ i )
-    {
+    {
```

```
         colStr[i] = strlen( (char *)strings[i] ) /
lenColCon;
         if(max < colStr[i])
             max = colStr[i];
@@ -368,7 +368,7 @@ int tabRow( void **strings, int
*bufType, int countColum, int lenColCon )
         {
             for (j = 0; j < (max + 1); ++ j)
             {
-                if((data[i][j] = (char *) malloc (lenColCon
* sizeof(char))) == NULL)
+                if((data[i][j] = (char *) malloc ( 2 *
lenColCon * sizeof(char))) == NULL)
                 {
                     printf("ERROR!\n");
                     exit(EXIT_FAILURE);
@@ -419,14 +419,42 @@ int tabRow( void **strings, int
*bufType, int countColum, int lenColCon )
         case 4:
             for( extraCounter = 0; extraCounter <=
colStr[i]; ++ extraCounter )
             {
+                int index = 0;
                 j = extraCounter * (lenColCon - 1);
-                for( offset = 0;
+                offset = 0;
+
+                while( ( ( lenColCon - 1 ) != index ) && (
( (char*)strings[i])[j] != '\0' ) )
+                {
+                    if( ( (char*)strings[i])[j] & 0x80 )
+                    {
+
+                        data[i][extraCounter][offset] =
((char *)strings[i])[j];
+                        ++ offset;
+                        ++ j;
+                        data[i][extraCounter][offset] =
((char *)strings[i])[j];
+                        ++ j;
+                        ++ index;
+                        ++ offset;
+                    }
+                    else
```

```
+                        {
+                            data[i][extraCounter][offset] =
((char *)strings[i])[j];
+                            ++ j;
+                            ++ offset;
+                            ++ index;
+                        }
+
+                    }
+
+
-                   for( offset = 0;
-                            ((offset != (lenColCon - 1)) &&
(((char *)strings[i])[j] != '\0' )); ++ offset, ++ j)
                    {
-                       data[i][extraCounter][offset] = ((char
*)strings[i])[j];
-                   }
                    /*Insert spaces*/
-                   for( offset; offset < (lenColCon - 1); ++
offset)
+                   for( index; index < (lenColCon - 1); ++
index, ++ offset)
                        data[i][extraCounter][offset] = ' ';
                }
                /*Insert spaces*/
```

После исправлений поля с символами кириллицы стали отображаться правильно (рис. 3.2).

```
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ ./quadratic-equation -a 1 -b 2 -c -3 --root1=1 --root2=-3
[RUN]: Запуск ./quadratic-equation
+-----------------------------------------------------------+-----------------------------------------------------------+
|Имя параметра                                              |Значение                                                   |
+-----------------------------------------------------------+-----------------------------------------------------------+
|Аргумент A                                                 |1                                                          |
+-----------------------------------------------------------+-----------------------------------------------------------+
|Аргумент B                                                 |2                                                          |
+-----------------------------------------------------------+-----------------------------------------------------------+
|Аргумент C                                                 |-3                                                         |
+-----------------------------------------------------------+-----------------------------------------------------------+
Сравнение эталонных и возвращаемых функцией корней
+-----------------------------------------------------------+-----------------------------------------------------------+
|Эталонные корни                                            |Корни, посчинанные функцией                                |
+-----------------------------------------------------------+-----------------------------------------------------------+
|1                                                          |1                                                          |
+-----------------------------------------------------------+-----------------------------------------------------------+
|-3                                                         |-3                                                         |
+-----------------------------------------------------------+-----------------------------------------------------------+
[PASS]: ./quadratic-equation : Тест пройден успешно
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$
```

Рис. 3.2

Также было выявлено что функция *tioHelp* не выводит название длинных ключей, если у них не существует аналогичных коротких.

Исправление, которое помогло решить эту проблему.

Теперь ключи отображаются правильно (рис. 3.3).

```
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ ./quadratic-equation --help
Использование: ./quadratic-equation [КЛЮЧ]... [ФАЙЛ]...
This is test
 -a <ПАРАМЕТР>                  Параметр A
 -b <ПАРАМЕТР>                  Параметр B
 -c <ПАРАМЕТР>                  Параметр C
 --root1 <ПАРАМЕТР>             Первый корень
 --root2 <ПАРАМЕТР>             Второй корень
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$
```

Рис. 3.3

Еще одна ошибка в логике была выявлена в функции *tioInit*. В случае, когда в программу вместе с командной строкой передавались неименованные параметры, они неправильным образом сохранялись в памяти кучи.

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**