Инструмент анализа структурного покрытия COVERest

Краткое руководство пользователя

Версия 1.4 от 11.02.2021

Аннотация

Данный документ содержит инструкцию по установке и работе с инструментом COVERest версии 3.20.6 для сбора и анализа структурного покрытия программного обеспечения (ПО). Инструмент ориентирован на применение его при разработке сертифицируемого авиационного ПО в соответствии с квалификационными требованиями КТ-178С и DO-178С [1, 2].

Содержание

1 НАЗНАЧЕНИЕ ПРОГРАММЫ

Инструмент COVERest предназначен для сбора структурного покрытия программ, написанных на языках C/C++, и выдачи отчетов о покрытии, которые могут использоваться при анализе полноты структурного покрытия, выполняемом для достижения целей, определенных в документе КТ-178С [1]. С помощью инструмента можно собирать покрытие структурных элементов по критериям MC/DC, DC и SC, а также покрытие связности компонентов ПО по данным и по управлению.

Для сбора информации о покрытии исходного кода ПО необходимо фиксировать прохождение потока управления программы через определяемые на этапе ее синтаксического анализа контрольные точки. Выбор контрольных точек производится в соответствии с типом собираемого покрытия - структурных элементов или связей, и критерием полноты покрытия, структурных элементов, который при разработке сертифицируемого авиационного ПО определяется его уровнем критичности в соответствии с [1].

Для обеспечения сбора структурного покрытия инструмент COVERest выполняет инструментирование программы, суть которого состоит в модификации ее исходного кода таким образом, чтобы обеспечить регистрацию прохождения потока управления через выбранные контрольные точки (трассировку программы), не изменяя ее основной функциональности.

Полученные в ходе выполнения программы трассировочные данные затем анализируются инструментом совместно с данными о структуре программы, сохраненными на этапе ее синтаксического анализа. В результате формируется отчет, содержащий информацию, необходимую при анализе структурного покрытия программы.

2 УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Для работы инструмента COVERest требуется наличие персонального компьютера с 64-битной архитектурой x86 (инструментальная среда) с предустановленной на ней Windows 10 x64 либо Debian 10 x64.

2.1Установка инструмента COVERest

2.1.1 Установка инструмента COVERest в среде ОС Windows

Инструмент **COVERest** поставляется в виде установочного пакета, реализованного как исполняемый файл *COVERest_install_vNNNN.exe, где NNNN – номер версии сборки.* При выполнении этого файла пользователю будет предложено определить имя и расположение корневой папки, в которой будет установлен инструмент (**Error! Reference source not found.**). По умолчанию корневая папка получит имя **`COVERest`** и будет расположена в папке, являющейся текущей при запуске установочного файла.

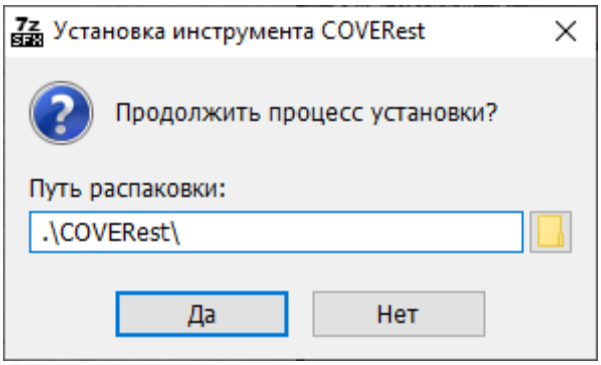

Рисунок 2.1 Установка инструмента

Структура корневой папке после установки инструмента следующая.

- Папка **bin** содержит исполняемые модули компонентов инструмента, здесь же содержатся динамически подключаемые библиотеки, необходимые для работы некоторых компонентов инструмента.

- Папка **demo** содержит демонстрационные примеры использования инструмента для сбора покрытия с формированием соответствующих отчетов о покрытии. Примеры демонстрируют возможности инструмента при сборе покрытия структурных элементов, связей по управлению и связей по данным компонентов специально подготовленного исходного кода макетных программ.

- Папка **supplementary** содержит программы сторонних производителей, которые могут понадобиться для обеспечения возможности функционирования инструмента и выполнения демонстрационных примеров.

- Папка **tracelib** содержит трассировочные библиотеки (.a) и включаемые файлы (.h), используемые при сборе покрытия в среде ОС Windows\Linux в разных режимах: при передаче трасс по UART в режиме с буферизацией, при передаче трасс по UART в потоковом режиме и при записи трасс непосредственно в файлы. Трассировочные библиотеки под Windows собраны с использованием компилятора и линкера tdm64-gcc-9.2.0.exe, установочный пакет которого включен в папку **supplementary.** Эта же версия пакета использовалась и при подготовке демонстрационных примеров.

Для обеспечения работы инструмента COVERest в Windows может потребоваться установка пакета *Microsoft Visual C++ 2019 Redistributable*. Файл дистрибутива ***VC_redist.x64.exe*** содержится в папке **supplementary**.

2.1.2 Установка инструмента COVERest в среде ОС Linux

Инструмент **E178Cover** для ОС Linux поставляется в виде архива COVERest_Linux_delivery_<номер версии>_<дата>.7z. Для установки инструмента требуется распаковать инструмент (например, в /home/user/COVERest).

Структура содержимого архива:

- Папка **bin/** содержит исполняемые модули – компоненты инструмента.

- Папка **demo/** содержит демонстрационные примеры использования инструмента для сбора покрытия с формированием соответствующих отчетов о покрытии. Примеры демонстрируют возможности инструмента при сборе покрытия структурных элементов и связей по управлению.

- Папка **lib/** содержит трассировочные библиотеки (.a) и включаемые файлы (.h), используемые при сборе покрытия в среде ОС Linux в разных режимах: при передаче трасс по UART в режиме с буферизацией, при передаче трасс по UART в потоковом режиме и при записи трасс непосредственно в файлы.

После установки необходимо запустить компонент **E178_CoverPrepare** и убедиться в совпадении версии установленного инструмента в строке вывода «COVERest tool ver. <версия инструмента>» и <версия инструмента> в заголовке исполняемого файла установочного пакета COVERest_Linux_delivery_<номер версии>_<дата>.7z

3 ВЫПОЛНЕНИЕ ПРОГРАММЫ

3.1 Компоненты инструмента COVERest

E178_CoverPrepare – синтаксический анализатор и инструментатор исходного кода. Входными данными (см. Рисунок 3.1) для работы является файл исходного кода и параметры инструментирования, а выходными данными - инструментированный файл исходного кода и вспомогательные файлы, необходимые для дальнейшей работы инструмента.

TraceClient – компонент, используемый для получения трассировочных данных, передаваемых по каналам с UART-интерфейсам.

E178_CoverAnalyze, E178_CCCAnalyze, E178_DCCAnalyze – генераторы отчетов о покрытии структурных элементов, связей по управлению и связей по данным. Входными данными (см. Рисунок 3.2) являются файлы с трассировочными данными (`.data`), полученными при прогоне тестов, файл с результатами синтаксического анализа инструментированного кода (`.dbm`), а выходными - отчеты о покрытии.

Трассировочная библиотека – необходима для работы инструментированного исходного кода.

irun – средство автоматизации включения инструментирования в регулярную процедуру сборки (без сбора покрытия).

Рисунок 3.1 Входные и выходные данные компонента E178_CoverPrepare

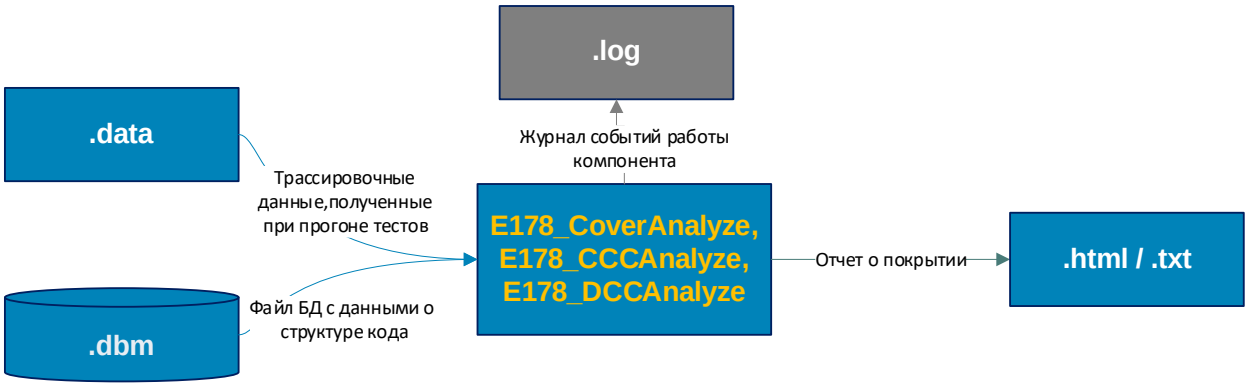

Рисунок 3.2 Входные и выходные данные компонентов E178_CoverAnalyze, E178_CCCAnalyze и E178_DCCAnalyze.

3.2 Настройка лицензии инструмента COVERest

При первом запуске инструмента без выпущенной лицензии инструмент выдаст ошибку об отсутствии лицензии и выведет на экран сигнатуру рабочей станции:

```
E178Cover tool ver. 3.20.6
[ERROR] License validation: invalid license
[INFO] Client signature = ABa4-begc-qgA=
```

Полученную сигнатуру необходимо передать поставщику инструмента для выпуска новой лицензии инструмента.

Полученный от поставщика файл лицензии pc_license.lic должен размещаться в директории запуска исполняемого файла инструмента.

После размещения корректной лицензии инструмент будет выдавать информацию о найденной лицензии:

```
E178Cover tool ver. 3.20.6
[INFO] License validation: license found
[INFO] License valid to 2020-12-20
```

3.3 Демонстрационные примеры для проверки работоспособности инструмента

3.3.1 Примеры сбора покрытия структурных элементов

Примеры 1 и 2, демонстрирующие выполнение сбора покрытия структурных элементов, расположены в папках ***demo\example1*** и ***demo\example2***.

В папке ***demo\example1*** содержится демонстрационный пример, иллюстрирующий использование инструмента для сбора и анализа покрытия структурных элементов специально подготовленной макетной программы. Пример включает три варианта сбора покрытия с использованием трех разных механизмов передачи данных о покрытии: с использованием файлов, с передачей трасс по UART в режиме с буферизацией и с передачей трасс по UART в потоковом режиме. Первый вариант применим, когда покрытие собирается на инструментальном компьютере, а два других варианта могут использоваться и в случаях, когда покрытие собирается не на инструментальном компьютере (на целевом).

Для демонстрации этих трех режимов сбора покрытия в примере подготовлены три папки с командными и конфигурационными файлами:

- **demo\example1\CoverageAnalysisFile**,
- **demo\example1\CoverageAnalysisComBuffer** и
- **demo\example1\CoverageAnalysisComStream** – соответственно.

Кроме того, в пример включена папка **demo\example1\NativeProcess**, содержащая процедуры подготовки и выполнения тестов без сбора покрытия. Предполагается (по легенде), что эти процедуры были преобразованы в процедуры сбора покрытия. Сравнение исходных и результирующих командных файлов из соответствующих папок, может облегчить понимание способов использования инструмента для сбора покрытия.

В папке **demo\example1\Src** содержится исходный код макетной программы, а в папке **demo\example1\tests** - тесты для верификации макетной программы.

В демонстрационном примере используются общедоступные компоненты из пакета GCC, а также библиотека автоматизации модульного тестирования ***CUnit***. Установочные файлы для этих компонентов содержатся в папке **supplementary**. Для установки пакета GCC следует использовать дистрибутив ***tdm64-gcc-9.2.0.exe*** (на ее использование настроены включенные в поставку примеры), отключив при установке проверку наличия более новых версий пакета.

В примере также используются файлы из папки **lib\Windows**, содержащей три варианта трассировочной библиотеки в соответствующих подпапках:

- **lib\Windows\file -** для передачи данных о покрытии с использованием файлов,
- **lib\Windows\buffer -** для передачи данных о покрытии по UART в режиме байтовых массивов и
- **lib\Windows\stream -** для передачи данных о покрытии UART в потоковом режиме.

Каждый из этих вариантов содержит подключаемый файл **trace_coverage.h** с определением интерфейса библиотеки и файл **libtrace.a** (библиотечный файл в формате GCC), содержащий ее реализацию.

Для выполнения примера необходимо последовательно запустить три командных файла из папки **demo\example1\CoverageAnalysis<вариант>** (где **<вариант>** — это **File**, **ComBuffer** или **ComStream**):

- **1_instrument_and_compile.cmd** – для инструментирования и компиляции исходного кода макетной программы;
- **2_build_and_run_tests.cmd** – для компиляции, сборки и выполнения тестов компонентов инструментированной программы;
- **3_create_report.cmd** – для получения отчета о покрытии.

В результате выполнения этих файлов будет создана папка с результатами покрытия **demo\example1\CoverageAnalysis\coverage_results**. Для их просмотра следует открыть в браузере файл **report.html** из этой папки.

Варианты примера, в которых данные о покрытии передаются по UART, настроены на использование нуль-модемного соединения между COM-портами, которые могут быть как реальными, так и виртуальными. При запуске командного файла **2_build_and_run_tests.cmd** можно указать порты и скорость передачи данных через них в параметрах командной строки, например задание параметров **COM1 COM2 19200** приведет к тому, что данные о покрытии будут передаваться от **COM1** к **COM2** со скоростью **19200** б/сек. Значения по умолчанию – **COM3 COM4 9600**.

В варианте примера из папки **demo\example1\CoverageAnalysisComBuffer** для приема данных о покрытии используется консольная программа TraceClient.

Анализируя полученный отчет о покрытии, можно обнаружить, что часть кода файла **calculate_led.c** не покрыта вследствие неполноты теста **Test_calculate_DD.c**. Для устранения

этой проблемы следует изменить строки 29-30 этого файла (расположенного в папке **demo\example1\tests\calculate_DD**) с

```
Calculate_DD(10000, 5000, TRUE, FALSE, TRUE, &leds);
CheckLedStateEquals(leds, TRUE, FALSE, FALSE);
```

заменив их содержимое следующим образом

```
Calculate_DD(10000, 5000, TRUE, TRUE, TRUE, &leds);
CheckLedStateEquals(leds, FALSE, FALSE, FALSE);
```

Это можно сделать или вручную, или запустив командный файл **4_update_test.cmd**, после чего можно сохранить для сравнения полученный отчет о покрытии, например, переименовав папку **coverage_results** (при выполнении файла **4_update_test.cmd** это будет сделано автоматически) и повторить шаги 2 и 3, т. е. запустить командные файлы **2_build_and_run_tests.cmd** и **3_create_report.cmd**. В результате этих действий в папке **demo\example1\CoverageAnalysis\coverage_results** появится новый отчет о покрытии, в котором файл **calculate_led.c** будет покрыт полностью.

Вернуть данные примера в первоначальное состояние можно, запустив командный файл **clean.cmd**.

Для удобства работы с примером в каждый из его вариантов дополнительно включены два командных файла: **run_initial.cmd** – обеспечивающий последовательный запуск шагов 1, 2 и 3, и **run_updated.cmd** - обеспечивающий запуск шагов 4, 2 и 3. Для работы примеров должна быть настроена переменная окружения TDMGCCPATH, указывающая путь к компилятору gcc. (Например, можно первой строкой файлов **run_initial.cmd** и **run_updated.cmd** добавить **set TDMGCCPATH=c:\TDM-GCC-64**).

При использовании передачи данных о покрытии по UART в обоих этих файлах можно указать параметры портов, с которыми на шаге 2 будет вызван соответствующий командный файл **2_build_and_run_tests.cmd**.

В папке ***demo\example2*** содержится более простой пример, в котором трасса записывается непосредственно в файл.

3.3.2 Примеры сбора покрытия связности по управлению

Примеры 3 и 4, демонстрирующие выполнение сбора покрытия связности по управлению, расположены в папках **demo\example3 и demo\example4**.

Пример 3 построен аналогично примеру 1 и в нем используется тот же самый исходный код, что и в примере 1. Также как и в примере 1, в нем реализовано три варианта сбора покрытия – с записью трассы в файл и с передачей трассы по UART в режиме с буферизацией и в потоковом режиме. Эти три варианта реализованы в папках, содержащих командные файлы для их выполнения с именами **demo\example3\ControlCouplingAnalysis<вариант>**, где **<вариант>** — это **File**, **Buffer** или **Stream**.

В примере 4 трасса записывается непосредственно в файл, а командные файлы для его запуска расположены в папке **demo\example4\ControlCouplingAnalysis**.

3.3.3 Примеры сбора покрытия связности по данным (только для Windows)

Примеры 5 и 6, демонстрирующие выполнение сбора покрытия связности по данным, расположены в папках ***demo\example5*** и ***demo\example6***. Трасса в этих примерах записывается непосредственно в файл.

4 СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Квалификационные требования. Часть 178С. Требования к программному обеспечению бортовой аппаратуры и систем при сертификации авиационной техники. АР МАК
2. RTCA/DO-178C. Software Considerations in Airborne Systems and Equipment Certification. RTCA Inc., 2011.

ПРИЛОЖЕНИЕ А. **Файлы инструмента**

`*.ini` – конфигурационный файл, содержащий в себе настройки для запуска *CoverPrepare.exe* и *irun.exe*.

`*.ins.c, *.ins.cpp` – инструментированные файлы исходного кода.

`*.dbm` – файл с результатами синтаксического анализа инструментируемого кода а также с объединенными трассировочными данными. Содержит в себе информацию обо всех объектах исходного кода, которые были инструментированы и должны быть покрыты, а также о результатах покрытия. Создается и пополняется инструментатором, используется генератором отчетов (анализатором).

`*.covmap.c` – файл, содержащий определения статических данных, используемых при регистрации событий прохождения потока управления через трассировочные точки в инструментируемом коде. Этот файл содержит информацию, общую для всех файлов, инструментированных с использованием одной и той же базы данных (БД) синтаксического анализа, и имеет имя, производное от имени файла этой базы данных

`*.data` – текстовый файл с информацией о прохождении потока управления. Формируется с помощью библиотеки `libtrace` при работе инструментированного кода.

`*.log` – файл с сообщениями об ошибках и предупреждениях, выдаваемых в ходе работы инструмента.