DOI: 10.15514/ISPRAS-2025-37(1)-1

Инструментарий для контроля качества тестирования функциональности и безопасности операционных систем реального времени

Аннотация. За последние несколько лет в популярный инструмент с открытым исходным кодом для сбора покрытия по коду llvm-cov были добавлены критерий MC/DC, поддержка встраиваемых систем и атомарные счётчики. Должно пройти некоторое время, прежде чем пользователи оценят эти возможности. В этом исследовании мы также предлагаем реализацию атомарного бинарного счётчика, чтобы обойти проблемы с использованием памяти во встраиваемой среде с низким объемом памяти, а также реализацию атомарного накапливающего счётчика для 32-разрядных SMP-систем, которые широко используются в современной индустрии, критически важной для безопасности. Мы выполняем серию экспериментов, демонстрирующих эффективность наших изменений.

Ключевые слова: llvm-cov, mc/dc, модифицированное покрытие условий решений, встраиваемая система, система реального времени, критерий тестового покрытия, критерий полноты тестирования, гонка по данным, состояние гонки.

Для цитирования: Инструментарий для контроля качества тестирования функциональности и безопасности операционных систем реального времени. Труды ИСП РАН, том 37, вып. 1, 2025 г., стр. xx–xx. DOI: 10.15514/ISPRAS–2025–37(1)–1.

Tooling for quality control, functionality and security testing of real-time operating systems

Abstract. MC/DC criterion, some support for embedded systems, and atomic counters were added to the popular open-source code coverage tool llvm-cov over the last few years. It will take some time for the users to evaluate these features. In this study we provide our feedback on using them for testing an industrial-grade project — a real-time embedded operating system. In this study we also propose llvm-cov atomic binary counter implementation, as well as atomic accumulative counter implementation for 32-bit SMP systems, which have common use in modern safety-critical industry. We perform a series of experiments demonstrating the effectiveness of our changes.

Keywords: llvm-cov, mc/dc, modified condition decision coverage, embedded system, real-time system, code coverage, coverage criteria, data race, race condition.

For citation: Tooling for quality control, functionality and security testing of real-time operating systems. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 1, 2025, pp. xx-xx. DOI: 10.15514/ISPRAS-2025-37(1)-1.

1. Введение

Измерение структурного покрытия по коду — это один из двух основных методов, используемых для оценки качества функционального тестирования. Вторым методом является измерение покрытия функциональных требований.

llvm-cov — это инструмент с открытым исходным кодом, встроенный в компилятор Clang. Он поддерживает основные метрики структурного покрытия кода, такие как покрытие строк и ветвей. Поддержка метрики покрытия MC/DC была добавлена в 2024 году с выходом релиза LLVM 18.1.0.

Критерий покрытия MC/DC [1, 2] был изначально определен в 1992 году в стандарте DO-178B [1], который регулирует разработку программного обеспечения (ПО) в области гражданской авионики. С того момента критерий MC/DC был принят во многие стандарты, регулирующие разработку ПО в различных областях. К ним, например, относятся аэрокосмический стандарт NPR 7150.2D, стандарт автомобильной промышленности ISO 26262, железнодорожный стандарт EN 50128, медицинский стандарт IEC 62304 и стандарт общей промышленности IEC 61508.

MC/DC обычно рекомендуется для ПО критического уровня, поскольку обеспечение полноты по этому критерию даёт высокое качество тестирования, но и существенно повышает его трудоёмкость.

Согласно первичному определению MC/DC, его можно рассматривать как набор критериев, включающий покрытие ветвей, покрытие условий и выполнение специального требования MC/DC о том, что каждое элементарное условие должно демонстрировать независимое влияние на исход всего составного условного выражения, в котором оно состоит. Хотя под MC/DC обычно неформально понимают только выполнение специального требования, но в данной работе под MC/DC мы будем понимать исходное определение, подразумевающее набор критериев полноты. Поэтому, например, проблемы непосредственно относящиеся к покрытию по ветвям мы также относим и к проблемам обеспечения MC/DC. Отметим также, что в инструменте llvm-cov измерение выполнения специального требования MC/DC необходимо выполнять одновременно с измерением покрытия ветвей.

«Masking» версия MC/DC позволяет преодолеть некоторые практические трудности, связанные с первоначальным определением MC/DC, в частности, измерение покрытия логических выражений с логическими операциями короткой логики. Эта форма MC/DC была принята индустрией гражданской авионики в 2001 году, когда был выпущен документ DO-248B [3]. Именно эта форма MC/DC реализована в инструменте llvm-cov.

Инструмент llvm-cov имеет встроенную поддержку встраиваемых систем. Ключевыми возможностями являются поддержка двоичных (бинарных) счётчиков для критериев полноты строк и ветвей, а также механизм выгрузки покрытия по запросу вместо выгрузки по завершению программы. Однако реальная практика показывает, что этого недостаточно для полноценного использования инструмента во встраиваемых системах. В данной работе мы дополняем их. В частности, мы реализовали возможность атомарной установки бинарных счётчиков, реализовали атомарные накапливающие счётчики для 32-битных платформ, а также предложили протокол выгрузки покрытия на инструментальную машину для SMP-систем.

Работа имеет следующую структуру. В разделе 2 мы перечисляем основные характеристики встраиваемых систем, которые мы подразумеваем. Далее в разделе 3 мы описываем имеющиеся возможности llvm-cov для встраиваемых систем. В разделе 4 мы реализуем несколько дополнительных возможностей в llvm-cov для встраиваемых систем. В разделе 5 мы измеряем производительность и потребление памяти в разных конфигурациях llvm-cov, включая реализованные нами возможности. Далее в разделе 6 мы выполняем обзор связанных работ и делаем выводы в разделе 7.

2. Особенности встраиваемых систем

В данной работе мы ограничиваемся рассмотрением встраиваемых систем, обладающих следующими характеристиками:

1. Они поставляются с микроконтроллерами, имеющими 32-разрядную или 64-разрядную архитектуру.
2. Они обрабатывают исключения.
3. Они поддерживают SMP (симметричную многопроцессорную обработку) с атомарными операциями.
4. Они могут обладать относительно небольшим запасом физической памяти.
5. Они могут поддерживать изоляцию памяти (например, с помощью MMU).
6. Они не имеют доступную для записи файловую систему.
7. У них есть внешние каналы связи, например UART, I2C или SPI.
8. При корректной работе они не предполагают своего завершения (выполняются в бесконечном цикле).

3. Устройство llvm-cov для встраиваемых систем

Инструмент llvm-cov имеет две реализации сбора покрытия по исходному коду:

- Первая реализация по своему устройству и формату сохраняемых данных покрытия совместима с инструментом gcov [4]. Данной реализацией поддерживаются метрики покрытия строк и ветвей, но не поддерживается MC/DC.
- Вторая реализация по принципу работы во многом схожа с первой, однако не совместима с ней. На данный момент в проекте LLVM активно развивают именно эту реализацию. Примечательно, что именно она поддерживает сбор покрытия по метрике MC/DC, поэтому для нас она представляет наибольший интерес. Далее речь пойдет именно об этой реализации.

Сбор покрытия по коду в инструменте llvm-cov для каждой поддерживаемой метрики устроен по одной схеме. Рассмотрим более подробно, как устроена схема его работы для встраиваемых систем (Рис. 1).

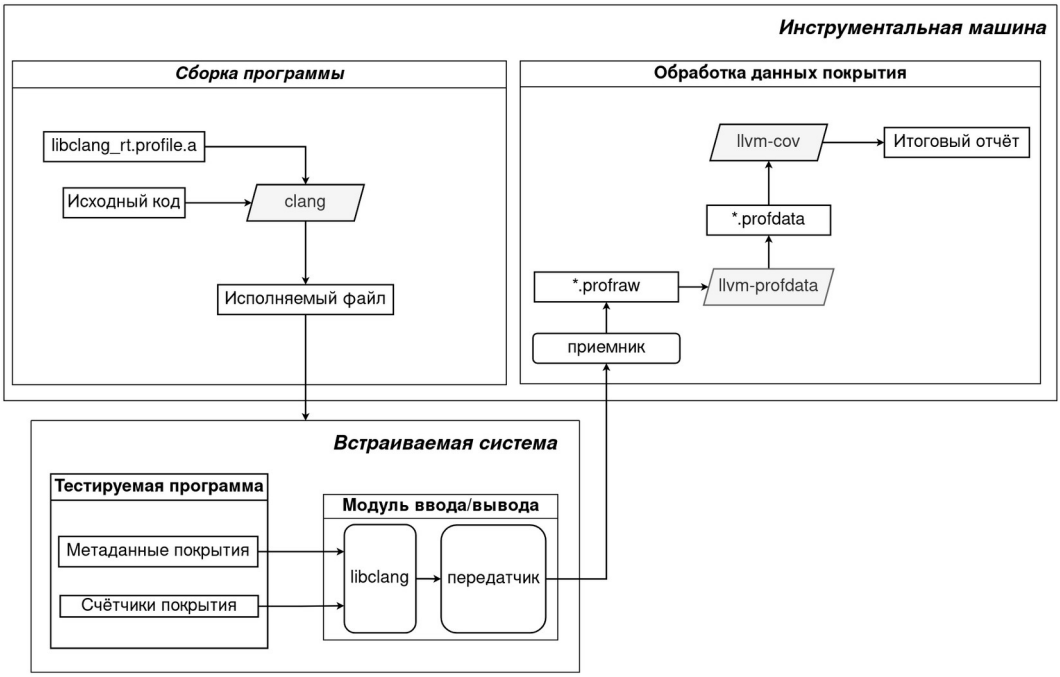

Рис. 1. Схема работы llvm-cov для встраиваемых систем

Fig. 1. llvm-cov workflow for embedded systems

3.1 Сборка программы

Прежде всего тестируемая программа компилируется и компонуется в бинарный файл, который далее будет запускаться на целевом устройстве.

Для компиляции программы со сбором покрытия используются следующие опции Clang: *-fprofile-instr-generate* и *-fcoverage-mapping*. При этом покрытие будет измеряться по строкам и ветвям. Для измерения покрытия по MC/DC необходимо использовать дополнительную опцию *-fcoverage-mcdc*.

Счётчики, использующиеся для покрытия строк и ветвей, имеют размер 8 байтов и обновляются с помощью инкремента, позволяя накапливать число выполнений соответствующих частей инструментируемой программы.

Для сбора покрытия по метрике MC/DC для каждого составного условного выражения (с несколькими элементарными условиями) создаётся битовый вектор, каждый элемент которого отражает покрытие соответствующего набора значений элементарных условий. Все такие битовые векторы объединяются в один общий массив. Для установления нужного бита идёт обращение по конкретному индексу этого массива.

Для инструментирования кода компилятором в бинарный файл программы добавляются различные секции со счётчиками или с метаданными покрытия. Среди них есть как загружаемые в память при выполнении

программы, так и незагружаемые. Из загружаемых можно выделить те секции, которые содержат счётчики покрытия:

- *__llvm_prf_cnts* — секция со счётчиками, собирающими покрытие по метрикам покрытия строк и ветвлений;
- *__llvm_prf_bits* — секция, представляющая из себя битовый массив, используемый для сбора покрытия по метрике MC/DC.

В обычном режиме инструментирования секции метаданных, загружающиеся в память при выполнении программы, используются при выгрузке данных покрытия в формате **.profraw*.

Однако Clang также поддерживает специальную опцию *--profile-correlate={debug-info, binary}*, при использовании которой метаданные сохраняются в некотором ином виде и не загружаются в память. Таким образом, из всех секций, относящихся к сбору покрытия, в память будут загружаться только необходимые секции со счётчиками. Это позволяет снизить количество потребляемой памяти на инструментирование, уменьшить размер исполняемых файлов, уменьшить объём данных о покрытии, передаваемых на инструментальную машину.

Для того, чтобы уменьшить влияние инструментации на общее поведение программы, в llvm-cov реализованы однобайтовые бинарные счётчики, принимающие только 2 значения – 0, когда соответствующий элемент кода покрыт, и 1 – в противном случае. Эти счётчики подключаются с помощью опции *-enable-single-byte-coverage=true*. Для их обновления используется одна 8-битная операция записи в память, что на большинстве аппаратных платформ выполняется быстрее операции инкремента.

Опция *-fprofile-update={single, atomic...}* позволяет сделать счётчики и операции для их обновления атомарными, что предотвращает возникновение гонок по данным в многопоточных программах.

Также при компоновке бинарного файла может использоваться библиотека компилятора, подключающая специальные функции, предназначенные для удобной выгрузки данных покрытия в условиях отсутствия файловой системы в используемой среде выполнения.

3.2 Выполнение на встраиваемой системе

Инструментированный код может начинать выполняться сразу после загрузки кода и данных в оперативную память без вызова функций-конструкторов, настраивающих метаданные покрытия. Во время выполнения счётчики обновляются. В любой момент пользователь может явно инициировать выгрузку информацию о покрытии на инструментальную машину.

3.3 Выгрузка покрытия на инструментальную машину

Процесс выгрузки данных покрытия происходит следующим образом.

1. Сначала пользователь должен вызвать функцию *__llvm_profile_get_size_for_buffer* для определения общего размера выгружаемых данных покрытия.
2. Далее пользователь самостоятельно выделяет буфер заданного размера.
3. Вызовом функции *__llvm_profile_write_buffer* все данные покрытия копируются в указанный в качестве параметра буфер.
4. После чего, используя доступные физические каналы, пользователь может передать содержимое данного буфера на инструментальный хост.

3.4 Составление отчёта о покрытии на инструментальной машине

Переданный на хост массив байтов следует записать в файл с расширением *.profraw*. Далее из этого файла с помощью утилит *llvm-profdata* и *llvm-cov* формируется итоговый отчёт о покрытии в удобном формате.

Также для формирования отчёта в HTML формате может быть использована утилита LCOV [5].

4. Адаптация llvm-cov к встраиваемым системам

4.1 Атомарные накапливающие счётчики

Накапливающие счётчики в llvm-cov используются для измерения покрытия строк и ветвей. Они имеют размер 64 бита на любой аппаратной платформе. Это необходимо для того, чтобы счётчики не переполнялись при многократном обновлении во время выполнения программы.

В llvm-cov атомарное обновление счётчиков по опции *-fprofile-update=atomic* работает ожидаемым образом только для платформ, поддерживающих 64-битные атомарные операции инкремента. В связи с тем, что для большинства 32-битных платформ такая поддержка обычно недоступна, то на них при использовании данной опции счётчики обновляются неатомарным образом. Тем не менее, проблема гонок по данным при обновлении счётчиков для этих платформ также актуальна и её необходимо решать.

Для решения этой проблемы мы предлагаем использовать подход, предложенный в реализации аналогичной опции в инструменте gcov [4], а именно с помощью 32-битной атомарной операции сложения обновить сначала нижнюю половину счётчика, после чего обновить верхнюю половину с учётом возможного переполнения нижней половины.

4.2 Атомарные бинарные счётчики

Опция компилирования *-fprofile-update=atomic* делает атомарными только накапливающие счётчики в текущей версии llvm-cov. Она не реализована для бинарных счётчиков, подключаемых опцией компилирования *-enable-single-byte-coverage=true*.

Мы реализовали установку атомарных бинарных счётчиков при помощи атомарной операции записи (store) с семантикой relaxed.

4.3 Оптимизация использования памяти при измерении MC/DC

Ранее упомянутая опция *--profile-correlate,* позволяющая для оптимизации использования памяти не загружать метаданные покрытия в память, также полезна в следующем аспекте. Использование данной опции приводит к изменению формата **.profraw* файлов, получаемых после выгрузки данных покрытия. А именно, в новом формате удаляются практически все метаданные (кроме заголовка), использующиеся для связывания счётчиков с соответствующими элементами исходного кода, что приводит к сильному уменьшению общего размера данных. Таким образом, при выгрузке данных покрытия на хост будет передаваться меньшее количество байтов, что поможет снизить накладные расходы на эту передачу.

Стоит отметить, что на момент написания данной работы в llvm-cov была проблема, связанная с отсутствием поддержки данной опции при измерении MC/DC. Эту проблему мы решили и отправили соответствующее исправление (патч) в проект LLVM.

4.4 Протокол выгрузки бинарных счётчиков и счётчиков MC/DC в SMP-системах

В SMP-системах возможна ситуация, когда во время выгрузки счётчиков покрытия одним из потоков другие потоки будут выполнять некоторый инструментируемый код и обновлять соответствующие счётчики покрытия. Это может привести к гонкам по данным, а также некорректному результату измерения покрытия, в котором покрытие разных участков кода не будет согласовываться друг с другом. Стоит также отметить, что процессорное ядро, занимающееся выгрузкой покрытия, также не должно обновлять при этом счётчики, а значит, код, выполняющий выгрузку, не должен инструментироваться llvm-cov.

Таким образом, для предотвращения такой ситуации необходимо, чтобы в процессе выгрузки покрытия не происходило выполнение инструментированного кода ни одним из процессорных ядер.

Будем считать, что выгрузку покрытия одновременно осуществляет ровно одно процессорное ядро. Это справедливо, поскольку копирование данных покрытия в буфер и выгрузка содержимого буфера на хост, как правило,

происходит последовательно, а не параллельно. Тогда остальные ядра на время выгрузки должны прекратить выполнение любого инструментированного кода. Для обеспечения такого условия предлагается действовать согласно следующему протоколу:

1. Выключить прерывания на текущем ядре, выполняющем выгрузку покрытия. Это необходимо, чтобы во время выгрузки не переходить в обработчик прерывания, код которого также может инструментироваться, а также для предотвращения смены контекста.
2. Поднять специальный флаг, сигнализирующий о начале выгрузки.
3. Отправить межпроцессорное прерывание всем остальным ядрам.
4. Подождать, пока ядра перейдут к обработке прерывания и приступят к выполнению специального неинструментируемого кода в обработчике до опускания флага.
5. Выгрузить покрытие.
6. Опустить флаг, тем самым разблокировав все ядра.
7. Разрешить прерывания на текущем ядре.

5. Производительность и потребление памяти

Тестирование проводилось с помощью теста, который был реализован как пользовательское приложение для ARINC 653 совместимой операционной системе реального времени CLOS [6]. Данный тест создает несколько потоков, каждый из которых выполняет несколько тысяч итераций в цикле, в котором происходит вычисление нескольких логических выражений. Потоки работают в едином адресном пространстве в режиме SMP и выполняют код одной и той же функции. Поэтому при измерении покрытия они параллельно обновляют одни и те же счётчики, причём это единственные данные, которые они разделяют.

Аппаратная платформа, на которой выполнялось тестирование, обладает 32-битной архитектурой и имеет 4 процессорных ядра. Время выполнения теста, а также суммарный размер секций кода и данных ядра ОС и пользовательского приложения в зависимости от разных типов инструментации представлены в таблице 1.

Тип инструментации	**Доступность в llvm-cov**	**Использование памяти, Кб**		**Время выполнения в зависимости от числа потоков, мс**				
		Без опции profile-correlate	**С опцией profile-correlate**	**1**	**2**	**4**	**8**	**16**
Код без	Да	406	406	667	668	669	670	670

инструментации								
Неатомарные накапливающие счётчики	Да	705	622	1204	2454	6077	6010	5994
Атомарные накапливающие счётчики	Нет	1192	1110	4593	11007	29398	33916	33924
Атомарные бинарные счётчики	Нет	647	561	1003	1042	1566	1487	1465
Неатомарные бинарные счётчики	Да	647	561	1003	1042	1566	1487	1465
Атомарные бинарные счётчики + неатомарные счётчики MC/DC	Нет	672	590	1386	1723	2629	2589	2587
Атомарные бинарные счётчики + атомарные счётчики MC/DC	Нет	705	618	1454	1510	2368	2302	2274
Неатомарные накапливающие счётчики + неатомарные счётчики MC/DC	Да	729	643	1570	3400	8776	8525	8638
Атомарные накапливающие счётчики + атомарные счётчики MC/DC	Нет	1249	1167	5077	12074	36433	36618	36602

Таблица 1: Потребление памяти и замедление системы в зависимости от разных типов инструментации

На основе полученных результатов мы можем сделать следующие выводы об измерении покрытия с помощью llvm-cov во встраиваемых системах 32-битной архитектуры:

1. Неатомарные бинарные счётчики замедляют выполнение на 96%.
2. Атомарные бинарные счётчики замедляют выполнение на 96%.
3. Тест с атомарными бинарными счётчиками выполняется на 70% быстрее, чем тест с неатомарными накапливающими счётчиками.
4. Тест с атомарными бинарными счётчиками выполняется на 94% быстрее, чем тест с атомарными накапливающими счётчиками.
5. Тест с атомарными бинарными счётчиками и атомарными счётчиками MC/DC выполняется на 68% быстрее, чем тест с неатомарными накапливающими счётчиками и неатомарными счётчиками MC/DC.
6. Неатомарные бинарные счётчики потребляют на 8% меньше памяти, чем неатомарные накапливающие счётчики.
7. Атомарные бинарные счётчики потребляют на 8% меньше памяти, чем неатомарные накапливающие счётчики, и на 46% меньше памяти, чем атомарные накапливающие счётчики.

Также по данной таблице можно определить влияние используемой опции *--profile-correlate* оптимизации на общее потребление памяти:

1. При использовании бинарных счётчиков с данной опцией общее потребление памяти в среднем ниже на 13%, чем без данной опции.
2. При использовании накапливающих счётчиков с данной опцией общее потребление памяти в среднем ниже на 10%, чем без данной опции.

При этом изменение размера передаваемых данных покрытия при использовании опции сильно зависит от инструментируемого кода и количества инструментируемых функций. В нашем случае при инструментировании более 2500 функций размер данных уменьшился с 95 Кб до 9 Кб для бинарных счётчиков, и с 130 Кб до 43 Кб — для накапливающих.

5.1 Адаптация llvm-cov к ARINC 653 совместимой ОСРВ

ARINC 653 [7] совместимые операционные системы имеют специфические особенности, которые также необходимо учитывать для успешного использования llvm-cov.

Согласно архитектуре ARINC 653, код операционной системы распределён между ядром операционной системы и всеми разделами, то есть также выполняется в контексте каждой прикладной программы. Из этого следует, что для измерения покрытия по всей операционной системе и пользовательским приложениям все адресные пространства должны

инструментироваться с помощью llvm-cov. При этом счётчики покрытия, относящиеся к коду в некотором адресном пространстве, также принадлежат данному адресному пространству, поэтому во время выгрузки покрытия необходимо обеспечить выгрузку покрытия из всех адресных пространств.

В стандарте ARINC 653 существует понятие холодного и горячего старта раздела, означающее процедуру, при которой происходит перезагрузка целого раздела. В результате чего блоки памяти заново инициализицируются некоторыми стартовыми значениями, которые использовались при первой загрузке раздела в память. Данный процесс осуществляется с помощью системного вызова SET_PARTITION_MODE [7]. Таким образом, если блок памяти, содержащий счётчики покрытия, при каждой перезагрузке раздела будет восстанавливать своё исходное состояние, мы каждый раз будем терять весь накопленный профиль покрытия кода раздела. Для того, чтобы этого избежать, необходимо, чтобы секции со счётчиками не инициализировались повторно при выполнении данного системного вызова.

Инструментация llvm-cov замедляет выполнение операционной системы, что может нарушить гарантированное операционной системой время выполнения в худшем случае. Чтобы избежать возможных проблем, связанных с этим, мы можем уменьшить частоту срабатывания системного таймера, например, заменив два такта на один. При этом необходимо убедиться, что это не повлияет на результаты теста.

6. Обзор связанных работ

В статье [10] решается задача измерения покрытия по MC/DC в таких крупных проектах, как ядро Linux, с целью соблюдения требований сертификации DO-178C в аэрокосмической индустрии. Основным результатом работы является измерение структурного покрытия по ядру Linux с отображением на строки исходного кода, даже с применением опций оптимизации. В статье также подробно описывается устройство llvm-cov и его ограничения, указываются проблемы, с которыми столкнулись авторы, а также предалагаемые ими решения.

Подробное описание подхода, используемого для реализации бинарных счётчиков можно найти в обсуждении [11]. Там же объясняется описываются причины выбора базовых блоков программы, которые будут инструментироваться при использовании бинарных счётчиков. Данный подход позволяет уменьшить количество используемых счётчиков и суммарный размер инструментального кода для их обновления, что позволяет оптимизировать накладные расходы на выполнение инструментального кода.

В работе [12] описывается опыт использования инструментов llvm-cov и gcov для сравнения получаемых результатов покрытия с целью выявления

ошибок и другого рода неисправностей в самих инструментах. Основной целью авторов является квалификация инструментов для возможности использования их для сертификации ПО на соответствие стандарту DO-178C. В работе подробно описывается методика запуска тестовых программ и сбора покрытия по ним. При этом сбор покрытия осуществлялся по метрикам покрытия строк, ветвей и MC/DC, что отличает эту работу от аналогичных. В результате инструменты были применены к программам из 41 пакета Debian, а найденные при тестировании ошибки инструментов переданы разработчикам.

Статья [13] также посвящена проблеме тестирования инструментов для сбора покрытия llvm-cov и gcov. В данной работе описывается другой подход, основанный на генерации случайных входных данных для широко используемых программ, а также на генерации новых программ для тестирования инструментов. В результате в обоих инструментах было найдено несколько десятков неисправностей.

7. Заключение

В данной работе были предложены следующие улучшения llvm-cov для повышения применимости инструмента во встраиваемых системах:

- Атомарные накапливающие счётчики для 32-битных платформ.
- Атомарные бинарные счётчики для предотвращения возникновения неожиданных эффектов, приводящих к гонкам по данным при обновлении счётчиков.
- Возможность использования опции *--profile-correlate* с измерением покрытия по MC/DC для уменьшения размера исполняемых файлов и уменьшения размера данных о покрытии передаваемых встраиваемой системой на инструментальную машину.
- Протокол для выгрузки атомарных бинарных счётчиков и атомарных счётчиков MC/DC, который дополняет механизм выгрузки покрытия по запросу, предоставляемый llvm-cov для встраиваемых систем.

Данные улучшения позволяют избежать возникновения гонок по данным во встраиваемых системах, а также снизить потребление памяти при инструментировании.

Мы реализовали на практике все предложенные улучшения в нашей локальной версии компилятора Clang для их использования в нашей системе непрерывной интеграции и развёртывания с целью контроля качества промышленной операционной системы реального времени CLOS. В дальнейшем мы планируем интегрировать данные изменения в основной репозиторий проекта LLVM.

Список литературы

[1]. RTCA/DO-178B. Software Considerations in Airborne Systems and Equipment Certification. RTCA SC-167 / EUROCAE WG-12, 1992.
[2]. A paper by one of the authors was deleted.
[3]. DO-248B. Final report for clarification of DO-178B. Software considerations in airborne systems and equipment certification, 2001.
[4]. Gcov code coverage analyzer. Available at: https://gcc.gnu.org/onlinedocs/gcc/Gcov.html, accessed 02.04.2026.
[5]. LCOV graphical front-end for gcov. Available at: https://github.com/linux-test-project/lcov, accessed 02.04.2026.
[6]. A paper by one of the authors was deleted.
[7]. Aeronautical Radio Inc. Avionics application software standard interface part 1 required services. ARINC Specification 653P1-2, 2005.
[8]. Larson P., Hinds N., Ravindran R., Franke H. Improving the Linux Test Project with Kernel Code Coverage Analysis. Linux Symposium, 2003.
[9]. Kedia S. P., Bhattacharjee A., Kailash R., Dongre S.. Coverage and Profiling for Real-Time Tiny Kernels. 10th IEEE International Conference on Computer and Information Technology, 2010, pp. 1926- 1931, DOI: 10.1109/CIT.2010.328.
[10]. W. Zhang, A. Oppelt, I. Jeon, M. Park, S. H. VanderLeest and C. Wolber. An Open-Source Structural Coverage Tool for DO-178C Compliance. 2025 AIAA DATC/IEEE 44th Digital Avionics Systems Conference (DASC), 2025, pp. 1-10, DOI: 10.1109/DASC66011.2025.11257174.
[11]. Savrun G. [RFC] Single Byte Counters for Source-based Code Coverage. Доступно по ссылке: https://discourse.llvm.org/t/rfc-single-byte-counters-for-source-based-code-coverage/75685, 14.12.2023.
[12]. W. Zhang, J. Jia, E. Yu, D. Marinov, T. Xu. DebCovDiff: Differential Testing of Coverage Measurement Tools on Real-World Projects. 025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2025, pp. 1083-1094, DOI: 10.1109/ASE63991.2025.00094.
[13]. Y. Yang et al. Hunting for Bugs in Code Coverage Tools via Randomized Differential Testing. 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), 2019, pp. 488-499, DOI: 10.1109/ICSE.2019.00061.