Санкт-Петербургский государственный университет

Технологии программирования

Группа 20.Б07-мм

Семенов Андрей Юрьевич

# Непрерывный режим инструмента покрытия кода в LLVM и GCC

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
доцент кафедры системного программирования СПбГУ, к.т.н. Ю. В. Литвинов

Консультант:
Володин Вадим Евгеньевич, ООО «Яндекс»

Санкт-Петербург
2022

# Оглавление

# Введение

Тестирование программного продукта является одним из важнейших компонентов разработки качественного программного обеспечения. Каждый программист должен самостоятельно уметь писать различные виды тестов для проверки качества кода, также существуют специалисты по тестированию кода, проверяющие все нюансы и крайние случаи. Код, покрытый тестами, позволяет программисту контролировать и делать более безопасным процесс реализации и внедрения в программный продукт новой функциональности, в случае чего быстро определяя и устраняя уязвимые области.

Для оценки уровня качества тестирования программного продукта необходимо выделить критерии, которые однозначно бы давали ответ на вопрос о качестве и полноте проведенной над тестами работы, такие критерии называются QA-метриками. Метрики позволяют однозначно определить и описать множество важных для программного продукта характеристик, таких как качество продукта, соответствие его заявленным пользователем требованиям, стабильность использования, определение уязвимых областей кода.

Одной из таких метрик является тестовое покрытие кода, то есть полнота покрытия написанного кода программного продукта, соответствие его нужным требованиям. Существуют несколько принципов оценки тестового покрытия:

- покрытие программного кода — основано на демонстрации того, какой процент строк программного кода был задействован во время тестирования, вычисляется как отношение количества покрытых тестами строк к количеству всех строк;

- покрытие требований — отражает полноту покрытия тестами требований к программному продукту, для данной оценки требования разбиваются на некоторые независимые единицы, каждая из которых либо покрыта тестом, либо нет, вычисляется по аналогичной предыдущему пункту формулой как отношение количе-

ства покрытых требований к количество всех требований;

- проверочное покрытие на основе данных потока управления.

В повышении тестового покрытия кода помочь UTBot[4] — инструмент компании «Huawei» для генерации модульных тестов, позволяющий максимизировать тестовое покрытие кода. UTBot автоматизирует порой рутинную, но необходимую работу тестирования, позволяя быстро сгенерировать тесты для покрытия строки кода, функции или содержимого всего файла. Таким образом, UTBot — быстрый способ максимизировать процент тестового покрытия кода и, следовательно, обеспечить значительное улучшение качества кода, легкости его сопровождения, ускорение поиска уязвимых и проблемных областей кода.

UTBot поддерживает компиляторы семейства LLVM и GCC и на данный момент корректно работает и выдает соотвествующий действительности результат о тестовом покрытии кода в случае безошибочного исполнения программы, в случае же ее ошибочного завершения (например выброшенный сигнал или исключение, ложное выражение в assert и так далее) информация о тестовом покрытии области кода, завершающейся ошибочно, не записывается совсем.

# 1. Постановка задачи

**Целью** работы является создание в UTBot корректного и точного тестового покрытия в случаях ошибочного завершения программ при компиляции с помощью LLVM и GCC.

Для достижения данной цели на осенний семестр были поставлены следующие задачи:

- Изучить возможные способы решения задачи для обоих компиляторов
- Провести эксперименты по сравнению скорости работы инструментов покрытия
- Выбрать для каждого из компиляторов оптимальный способ, позволяющий сохранять информацию о покрытии кода даже в случае ошибочного завершения программы

На весенний семестр:

- Улучшить покрытие предложенным способом в UTBot для LLVM и GCC

# 2. Обзор

В данном разделе представлены обзор UTBot, подходов и инструментов записи покрытия кода, позволяющих решить поставленную задачу в LLVM и GCC.

## 2.1. UTBot

UTBot поддерживает работу с GCC и LLVM, помогая программисту покрыть код тестами и максимизировать покрытие. UTBot может генерировать тесты для покрытия строки, функции или файла. UTBot выводит данные о покрытии следующим образом: происходит создание отчета о покрытии с помощью компиляции и получения файла с информацией о покрытии .profdata (LLVM, Clang) или .gcda (GCC), после чего этот файл в формате JSON передается в обработку UTBot и происходит представление результата об отчете в IDE: если рядом со строкой стоит зеленый цвет — она покрыта тестом, если красный — не покрыта. Для получения отчета о покрытии в UTBot используются llvm-cov для LLVM, Clang и Gcov для GCC.

## 2.2. Инструменты

Для реализации данной задачи можно использовать следующие инструменты:

- Инструменты покрытия самих компиляторов
- Внешние инструменты, позволяющие получить данные о покрытии

### 2.2.1. llvm-cov

Для получения информации о покрытии кода в Clang можно использовать инструмент LLVM *llvm-cov* [2]. Для этого нужно:

1) скомпилировать файл с параметрами -fprofile-instr-generate -fcoverage-mapping;
2) создать переменную окружения LLVM_PROFILE_FILE и присвоить ей имя файла отчета для формирования и записи отчета в этот файл;
3) слить отчеты о покрытии в один файл с помощью llvm-profdata merge, получив требуемый файл в формате .profdata.

Информация о покрытии может быть представлена в человекочитаемой форме с помощью команды llvm-cov show или получена в формате JSON.

Получение отчета таким образом не решает проблему отстутствия информации о покрытии в случае ошибочного завершения программы. Эту проблему может решить continuous mode — формат записи отчета о покрытии, позволяющий получить его даже в случае ошибочного завершения программы. Continuous mode основан на идее записи на память отчета о покрытии с помощью функции mmap. Для его активации нужно компилировать с двумя дополнительными параметрами -mllvm -runtime-counter-relocation и при присвоении в переменную окружения имени файла отчета в начале добавить ”%c”. Воздействие continuous mode на отчет о покрытии можно видеть на рисунке 1.

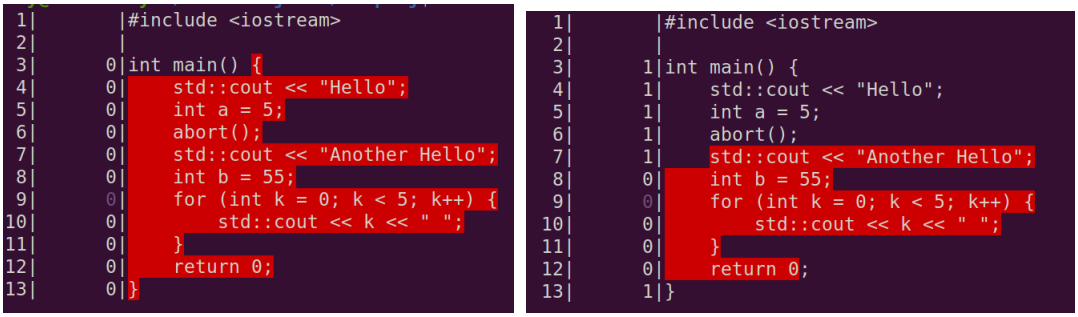


Рис. 1: Покрытие в случае ошибочного завершения программы: слева - без continuous mode, справа - с ним

### 2.2.2. Gcov

Для получения информации о покрытии кода в GCC можно использовать инструмент компилятора *Gcov* [1]. Для этого нужно:

1) скомпилировать файл с параметрами -lgcov -coverage;

2) запустить исполняемый файл.

Далее можно получить отчет о покрытии с помощью команды gcov с нужными параметрами (подцветка строчек цветом, например) и именем файла отчета в формате .gcda.

### 2.2.3. kcov, bcov

Существуют также альтернативные внешние инструменты для получения отчета о покрытии кода: *bcov* и *kcov*[3]. Они работают аналогичным образом (так как изначально kcov — форк bcov, его расширенная версия): после компиляции используется debug информация объектных файлов в формате DWARF. Сам процесс получения покрытия основан на идее расставления точек останова и далее использования инструментов debug. Данный способ позволяет получить информацию о покрытии кода в случае ошибочного завершения программы. Стоит отметить, что данные инструменты покрытия ограничены и не являются кроссплатформенными, но UTBot, который рассматривается в рамках данной задачи, также не является кроссплатформенным, работая, как и kcov, только на Linux.

# 3. Ход работы

В данном разделе представлено сравнение и обоснование выбора инструмента покрытия и план действия по реализации решения.

## 3.1. Сравнение инструментов для решения задачи и получения покрытия

Чтобы получить точное и оптимальное покрытие существуют два возможных варианта:

1) использование llvm-cov и continuous mode для LLVM и преобразование файла отчета о покрытии в GCC;

2) использование kcov для получения покрытия в обоих компиляторах.

В ходе работы было проведено изучение каждого из двух способов и сравнение способов между собой.

### 3.1.1. Использование инструментов компилятора

Для получения отчета о покрытии в случае LLVM можно использовать llvm-cov с активированным continuous mode, но проблема была в следующем: покрытие удавалось корректно записать в случае ошибочного завершения программы путем вызова функции std::abort(), в ряде других случаев (например, вызов исключения или сигнала; функции, которые могут вызвать std::abort(), такие как assert) покрытие записывалось некорректно, помечая покрытыми строки после завершения программы. Данный вопрос был тщательно исследован на примере функции assert, хотелось понять, что происходит между assert и самим вызовом std::abort(), что могло бы ломать корректную запись покрытия после завершения программы assert. Было выяснено, что assert вызывает в assert.h функцию __assert_fail(), которая реализована в библиотеке glibc и, судя по реализации, действительно вызывает std::abort(). Проблема оказалась обозначена в явном виде и заключалась как раз в том, что continuous mode не гарантирует корректную работу во всевоз-

можных ситуациях (а именно в случаях непредсказуемых изменений в потоке управления или в случае раскручивания стека при наличии исключений).

В ходе исследования GCC и инструмента Gcov было выяснено, что записать корректное тестовое покрытие в случае ошибочного завершения программмы невозможно, поэтому требуется использовать другие инструменты записи покрытия.

### 3.1.2. Использование внешних инструментов

Изначально была гипотеза о том, что kcov работает значительно медленее своих «оппонентов», встроенных в компиляторы, поэтому его использование откладывалось как рабочий, но запасной вариант, первым приоритетом выходило исследование методов получения корректного отчета о покрытии инструментами компилятора и корректности их работы. После выяснения факта о негарантированности работы continuous mode во всех ситуациях ошибочного завершения программы, было принято решение обратить внимание на kcov, исследовать его скорость работы по сравнению с llvm-cov и gcov. Гипотеза оказалась неверной, и оказалось, что kcov работает незначительно медленнее своих аналогов, что можно увидеть в таблицах 1 и 2. Эксперименты проводились на двух программах, количество замеров составляло от 12 до 15. Первая программа содержала вывод в консоль стандартным потоком, объявление переменной типа int, цикл на 100 итераций с инкрементированием объявленной ранее переменной, ещё один вывод в консоль. Вторая программа представляла из себя проект, состоящий из 2 файлов (main и SimpleClass), в main был цикл на 2000000 элементов, который вызывал вычисление функции, в которой создавался объект класса и свойству которого присваивалось вычисленное значение функции.

| Замер на программе № | Инструмент покрытия | Среднее ± среднеквадратическое отклонение, с |
|---|---|---|
| 1 | kcov | 0.757±0.012 |
| 1 | llvm-cov | 0.707±0.015 |
| 2 | kcov | 3.129±0.055 |
| 2 | llvm-cov | 3.046±0.016 |

Таблица 1: Результаты замеров для LLVM, Clang

| Замер на программе № | Инструмент покрытия | Среднее ± среднеквадратическое отклонение, с |
|---|---|---|
| 1 | kcov | 0.758±0.012 |
| 1 | Gcov | 0.672±0.008 |
| 2 | kcov | 2.570±0.055 |
| 2 | Gcov | 2.655±0.010 |

Таблица 2: Результаты замеров для GCC

## 3.2. План реализации решения задачи в UTBot

В случае GCC запись покрытия с помощью инструмента компилятора Gcov не может быть реализована, в GCC требуется использовать внешние инструменты, в случае LLVM и llvm-cov запись покрытия может быть получена, но она не является корректной во всех случаях ошибочного завершения программы. Kcov работает хорошо для обоих компиляторов, выдает корректное покрытие в случае ошибочного завершения программы, но исполняемый файл, сформированный для его использования, и отчет являются тяжеловесными. В силу приведенных выше рассуждений, то есть невозможности записи корректного покрытия кода в случае ошибочного завершения программы, был выбран инструмент записи покрытия kcov для обоих компиляторов. kcov

генерирует отчет в формате XML, поэтому для получения информации о покрытии было решено использовать библиотеку pugixml. Её плюсом являются легковесность и простота внедрения.

## 3.3. Реализация

### 3.3.1. Зависимости и docker образ

В первую очередь нужно было добавить kcov и pugixml в docker образ UTBot. Для работы UTBot используется дистрибутив Ubuntu версии 18.04, для него пакет pugixml изначально существует и доступен, пакет kcov доступен, начиная с 20 версии дистрибутива. Можно было либо собрать kcov из исходников, либо перейти на версию 20.04 дистрибутива Ubuntu, был выбран последний вариант. Таким образом, был обновлен docker образ (добавлены инструменты kcov и pugixml) и обновлена версия Ubuntu.

### 3.3.2. Генерация отчета

Генерация отчета инструментом kcov происходит с помощью предварительно скомпилированного с флагом -g исполняемого файла командой kcov *path_to_put_report path_to_executable*. UTBot генерирует MakeFile для запуска тестов, в build команду нужно было добавить флаг -g для корректной работы kcov, в run команде нужно было было заменить запуск исполняемого файла на kcov команду, запускающую этот файл и генерирующую отчет.

### 3.3.3. Получение информации о покрытии из сгенерированного отчета

Отчет о покрытии помещался в build папку в генерируемую папку report, в которой хранится файл cov.xml с отчетом о покрытии. Для получения информации о покрытии из этого файла была использована библиотека pugixml в силу легковесности и легкости интегрирования в проект.

В отчете в тегах сначала передается имя проекта, для которого отчет был сгенерирован, далее тег, отвечающий за имя файла, далее тег *lines*, содержащий в себе множество тегов *line*, которые содержат номер строки и количество раз, которые данная строка была исполнена.

С помощью pugixml информация о покрытии сохраняется и передается в дальнейшую обработку в специальной map CoverageMap, которая ключом имеет путь до файла типа *fs::path*, а значением структуру FileCoverage, содержащую в себе множества покрытых/непокрытых строк, множества покрытых/непокрытых подряд идущих отрезов кода.

# Заключение

В ходе выполнения данной работы были выполнены поставленные задачи:

- Были изучены потенциальные методы и инструменты для решения задачи
- Был проведен сравнительный эксперимент по замеру скорости времени получения отчета о покрытии с помощью kcov и инструментов компилятора llvm-cov и Gcov и выбран инструмент для получения информации о покрытии
- Реализован прототип инструмента, использующий kcov для генерации и получения информации о покрытии в UTBot

Исходный код доступен в репозитории на сервисе github[1].

[1] https://github.com/AndersenSemenov/UTBotCpp/tree/kcov_updated

# Список литературы

[1] Documentation of GCC instruments to get code coverage. — Access mode: https://gcc.gnu.org/onlinedocs/gcc/Gcov.html (online; accessed: 2021-12-26).

[2] Documentation of LLVM instruments to get code coverage. — Access mode: https://clang.llvm.org/docs/SourceBasedCodeCoverage.html (online; accessed: 2021-12-26).

[3] Kcov repository and documentation. — Access mode: https://github.com/SimonKagstrom/kcov (online; accessed: 2021-12-26).

[4] UTBot source code. — Access mode: https://github.com/UnitTestBot/UTBotCpp (online; accessed: 2021-12-26).