Слайд 1

Темой моей выпускной квалификационной работы является разработка программного модуля для анализа программ на языках C и C++ на недекларированные возможности.

Цель: Унификация и ускорение проведения сравнительного анализа статических и динамических трасс программ, написанных на C/C++

Задачи:

- исследование предметной области;
- сравнительный анализ существующих программных решений;
- выбор языка и среды разработки;
- разработка схемы данных ПМ АПНДВ;
- разработка схемы алгоритма ПМ АПНДВ;
- реализация ПМ АПНДВ;
- отладка и тестирование ПМ АПНДВ;
- разработка документации к ПМ АПНДВ;

Слайд 2

На текущий момент проферка программ на НДВ органом сертификации происходит вручную:

- с помощью специального ПО проводят статический анализ исходных кодов программного проекта;
- с помощью отладчиков, профилировщиков или эмуляторов проводят динамический анализ исполняемого файла, сохраняя трассы выполнения;
- данные статического и динамического анализа приводятся к общему виду;
- резрабатывается скрипт сравнения результатов статического и динамического анализа;
- с помощью скрипта сравнения ищутся несовпадения или подтверждается их отсутствие.

В свою очередь, ПМ АПНДВ умеет автоматически проводить действия, требуемые для проверки ПО, возращая оператору отчет, на основе которого оператор ПМ АПНДВ делает заключение о наличии или отсутствии НДВ в тестируемом ПО.

Слайд 3-4

Сейчас на рынке не существует программных решений, аналогичных ПМ АПНДВ, поэтому ПМ АПНДВ сравнивался функционал статических и

динамических анализаторов, а наиболее подходящее по критериям ПО становилось частью ПМ АПНДВ.

Слайд 5

В качестве языка программирования был выбран Nim, как язык с хорошим балансом между скоростью работы, легкостью написания кода и портируемостью. На развитие синтаксиса Nim повлиял язык программирования python – очень удобный в написании программ интерпретируемый язык, чьим большим минусом является скорость выполнения написанных на нем программ. Nim же компенсирует данный недостаток компиляцией в машинный код, через промежуточную компиляцию в код Си, что позволяет ему пользоваться всей стабильностью и высоким уровнем оптимизации, который предоставляют такие компиляторы, как gcc или clang.

Слайд 6

В качестве среды разработки был выбран текстовый редактор Vim. Vim не требователен к ресурсам, в отличие от большинства других популярных редакторов кода и IDE, имеет большую библиотеку плагинов, позволяющих идеально настроить окружение под себя. Разделяя работу с текстом на режим ввода и режим команд, Vim дает возможность разработчику быть максимально производительным, так как сложные действия по редактированию текста описываются нажатием буквально двух кнопок.

Слайд 7

Схема данных отражает операции, происходящие с данными во время работы ПМ АПНДВ. От оператора программа получает папку с мейкфайлом и именем программы, после чего собирает программу из исходников, проводит статический, динамический и сравнительный анализ, выводя оператору отчет о проделанной работе.

Слайд 8

Схема алгоритма отражает саму работу программы. Первым идет этап сборки, от которого зависит как статический, так и динамический анализ. После окончания сборки параллельно запускаются этапы статического и динамического анализа, а по завершению их обоих проходит сравнительный анализ.

В статическом анализе строится дерево вызовов программы, основываясь на том, как вызовы записаны в исходниках. После чего дерево переводится во

внутренний формат, и к нему добавляется информация времени компиляции, связанная с адресами функций в исполняемом файле.

Этап динамического анализа предваряет бинарный анализ, в котором ищется первая инструкция в сегменте кода программы и на нее выставляется программная точка останова. После этого ПМ АПНДВ собирает динамические трассы программы, вместе с мета-информацией о вызовах, и последним шагом преобразует собранную информацию во внутренний формат хранения данных.

Слайд 9

Графический пользовательский интерфейс минималистичен и отражает собой простоту запуска программы.

Слайд 10

Разработка ПМ АПНДВ проводилась с использованием техники Test-driven development, которая предусматривает написание тестов для кода перед тем, как разработчик напишет сам программный код, что помогает продумывать интерфейсы взаимодействия между элементами программы. В рамках данной техники было написано 43 модульных теста.

Слайд 11

Доклад про разработку coreutils был напечатан в сборнике докладов МИЭТ.