Методы анализа программного обеспечения

Методы анализа

Статический анализ кода - это процесс выявления ошибок и недочетов в исходном коде программ. Статический анализ можно рассматривать как автоматизированный процесс обзора кода (code review).

Динамический анализ кода - это способ анализа программы непосредственно при ее выполнении.

Статический анализ: преимущества

- Может использоваться на ранних этапах жизненного цикла программного обеспечения, прежде чем код готов для исполнения и до начала тестирования.
- Статические анализаторы проверяют даже те фрагменты кода, которые получают управление крайне редко.
- Низкие стоимостные затраты (например, нет необходимости создавать тестовые программы); разработчики могут запускать свои собственные виды анализа.

Статический анализ: недостатки

- Поскольку во время статического анализа делается попытка предсказать поведение программы, то иногда обнаруживается "ошибка", которой фактически не существует – это так называемое "ложное срабатывание" (false positive).
- Статический анализ, как правило, слаб в диагностике утечек памяти и параллельных ошибок.

Динамический анализ: преимущества

- Редко возникают "ложные срабатывания" – высокая продуктивность по нахождению ошибок.
- Для отслеживания причины ошибки может быть произведена полная трассировка стека и среды исполнения.

Динамический анализ: недостатки

- Полнота анализа ошибок зависит от степени покрытия кода. Кодовый путь, содержащий ошибку, должен быть обязательно пройден, а в контрольном примере должны создаваться необходимые условия для создания ошибочной ситуации.
- Происходит вмешательство в поведение системы в реальном времени. Это не всегда приводит к возникновению проблем, но об этом нужно помнить.

Инструменты статического анализа

“96 (91) C Static Analysis Tools” https://analysis-tools.dev/tag/c

5. cppcheck (4)
9. PVS-studio (10)

Что посмотреть еще

- gcc -c -fanalyzer имя_файла (>= 10)
- clang --analyze –Xanalyzer -analyzer-output=text имя_файла (>= 15)

Инструменты статического анализа, использующие clang

- scan_build
- CodeChecker
- clang-tidy

Инструменты динамического анализа

- valgrind
- санитайзеры
- gcov
- gdb
- gprof

Address sanitizer

Обнаруживаемые ошибки:

- Выход за пределы локального/глобального/динамического массива.
- Неверное использование локальных переменных.
- Неверная работа с динамической памятью.
- Утечки памяти.

Address sanitizer

clang (>= 3.1), gcc (>= 4.8)

Ключи (для clang)

-fsanitize=address

Чтобы выдача была читаемой

-fno-omit-frame-pointer -g

Memory sanitizer

Обнаруживаемые ошибки:

- Использование не инициализированных локальных и «динамических» переменных.

Ключи (для clang)

```
-fsanitize=memory -fPIE -pie
```

Чтобы выдача была читаемой

```
-fno-omit-frame-pointer -g
```

Undefined behavior sanitizer

Обнаруживаемые ошибки:

- Различные виды неопределенного поведения.

Ключи (для clang)

-fsanitize=undefined (можно указать «подмножество»)

Чтобы выдача была читаемой

-fno-omit-frame-pointer -g

Memory sanitizer: точность работы

Uninitialized values occur when stack- or heap-allocated memory is read before it is written. MSan detects cases where such values affect program execution.

... It will tolerate copying of uninitialized memory, and also simple logic and arithmetic operations with it. In general, MSan silently tracks the spread of uninitialized data in memory, and reports a warning when a code branch is taken (or not taken) depending on an uninitialized value.

It is critical that you should build all the code in your program (including libraries it uses, in particular, C++ standard library) with MSan.

Инструменты динамического анализа: gcov

gcov – утилита для исследования покрытия кода.

gcov предоставляет информацию о том, сколько раз исполнился во время работы программы каждый участок кода. Утилита позволяет создать так называемый «аннотированный» листинг исходного кода программы, который содержит информацию о частоте выполнения каждой строки.

gcov обрабатывает программы, которые были получены только с помощью компилятора gcc.