Д. С. Панкевич

SELF-MODIFYING CODE

Рассматривается альтернатива использования механизма самомодификации кода.

Введение

Считается, что выполняющаяся компьютерная программа отличается от данных, которые она обрабатывает. Однако это отличие размывается, когда компьютерная программа модифицирует сама себя. Модифицированная компьютерная программа затем выполняется как часть исходной программы.

SMC (Self-modifying code) - это изменение того что делает программа в режиме реального времени. В общем смысле, если программный код не делает одни и те же вещи при каждом его исполнении, то это и есть самомодифицирующийся код. SMC встречается во всех языках программирования в том или ином проявлении. В языках программирования высокого уровня есть множество различных механизмов, позволяющих достичь этого, к примеру: циклы, условные операторы, полиморфизм и т.д.

Однако есть ещё один тип самомодифицирующегося кода — это код, который изменяет свои собственные команды во время работы. Исполнение такого кода чаще всего ассоциируется с хакерским вмешательством, так как он усложняет исследования кода. Тем не менее, его можно применять и в легальных целях как инструмент, позволяющий достигать вещей, не доступных языкам высокого уровня из-за всяческого запрета такого подхода со стороны операционной системы.

По времени проведения модификации метод делится на:

- Модификация при инициализации – проводится один раз, перед запуском изменяемого кода;
- Модификация на лету (on-the-fly) – изменение состояния программы во время исполнения.

В обоих случаях изменение проходит непосредственно в машинном коде

I. Принцип работы SMC

В отведенной на работу программы области памяти выделяют 3 основных сегмента:

- сегмент данных;
- сегмент стека;
- сегмент кода.

Все эти сегменты расположены в каком-то месте в RAM, и все они принадлежат страницам. Операционная система маркирует эти страницы различными правами доступа. Так некоторые страницы позволяют читать и записывать информацию, некоторые позволяют исполнять себя и т.д. По умолчанию страница сегмента кода имеет права доступа чтения и исполнения, и операционная система заблокирует попытку вмешательства в сегмент кода на его изменение или перезапись. И, таким образом, имеется возможность только читать данные с этой страницы, либо исполнять данные на этой странице. В чём же хорош язык ассемблера так это в том, что он позволяет вручную разметить сегменты памяти и так же наделить их необходимыми правами доступа. Вследствие чего появляется возможность получить доступ к модификации кода.

II. Как работать с SMC

SMC как таковой в полной мере доступен только на языках ассемблера. Однако его также можно использоваться как ассемблерные модули в многофайловых проектах программ высокого уровня.

Использование SMC подразумевает обладание возможностью оперирования кодом как данными. Для этого необходимо странице, где расположен сегмент целевого модифицируемого кода, предоставить права на запись. Делается это в объявлении сегмента кода. Таким образом появляется возможность изменять содержимое бит кода в режиме работы реального врмемени.

Модифицировать код можно как из самого модифицируемого сегманта кода, так и из других сегментов кода. Модификация кода обычно происходит в рамках блока какой-либо части подпрограммы. Чтобы безопастно оперировать кодом как данными, необходимо соблюсти следубщие выведенные правила:

1. Размер модифицируемого блока должен быть больше или равен размеру записываемых машинных команд.

2. Модифицируемый блок не должен содержать в себе непарное количество инструкций работы со стеком.

3. После модификации кода для сохранения его непрерывности и целостности необходимо дополнить блок модифицируемого кода машинными кодами 90h, они же NOP (No Operation) – инструкции процессора, которая не делает ничего. Количество требуемых NOP равно разнице длин суммы машинных кодов модифицируемого блока кода и суммы машинных кодов записываемого кода в байтах.

Модификация кода происходит путём записи данных по адресу. Чтобы явно указать в коде ассемблера с какого момента необходимо начать проводить запись можно использовать метки. Метки – мнемоники ассемблера, которые сами по себе являются виртуальным адресом команды или данных. С их помощью можно «заключить» фрагмент интересующего кода в блок, тем самым во первых имея возможность указать место, с которого будет производится запись, а во вторых узнать размер машинных команд в блоке, посчитав разницу адресов конца и начала блока.

III. Достоинства и недостатки SMC как подхода к написанию кода

SMC можно использовать для уменьшения размеров объектного кода. Так, к примеру, в коде алгоритма построения прямых Брезенхема можно уменьшить количество кода практически в 2 раза, определив по входным данным выбор оси инкремента и соответсвенно модифицировав инструкцию инкремента цикла, избавившись от дублирования основной части алгоритма.

Уменьшение кода это один из возможных подходов использования механизма SMC. Наиболее значимым являетя перспектива его применения в области алгоритмизации. SMC из-за своей абсолютной гибкости позволяет составлять с его использованием алгоритмы, которые подстраиваются под решение задачи во время выполнения. Или позволит составлять алгоритмы, которые составляют другие алгоритмы, которые в свою очередь выполняют требуемую задачу. И в данном случае у алгоритма всё не ограничевается набором возможных действий «на выбор», ведь алгоритм составляется из элементарнейших частей – машинных комманд, которые в своих комбинациях имеют неограниченную вариативность в действиях для достижения желаемого результата.

SMC позволяет делать в режиме реального времени обновление программного обеспечения, транслировать байт-код в машинный код непосредственно во время работы программы для увеличения производительности.

Можно так же применять самомодификацию кода в критичных к скорости местах для ускорения работы. Так, например, во время исполнения можно уменьшить длину критического пути исполнения. Вместо установки и последующей многократной проверкой флагов с условными переходами можно всего лишь изменить адрес и тип перехода в машинном коде.

К недостаткам SMC в программировании следует отнести отсутсвие безопастного обращения с кодом, сложности с разработкой, тяжесть в длительной поддержке программного продукта, ограниченность в размерах целевых блоков кода. Последнего можно избежать использованием динамической компиляции.

Современные языки программирования и среды разработки с их компиляторами или трансляторами не позволяют раскрыть весь потенциал самомодифицирующегося кода. Языки Perl, PHP и Python позволяют программе создавать новый код во время выполнения и выполнять его, но не позволяют самомодифицироваться существующему коду. Существуют аналогичный механизм модфикации работы программы, который достигается путём изменения указателей на функции. В действительности при таком подходе машинный код не изменяется.

В наше время самомодификация кода применяется в технологии JIT-компиляции (Just-in-time compilation). Эта технология позволяет увеличить производительность программных систем, использующих байт-код, путём компиляции байт-кода в машинный код или в другой формат непосредственно во время работы программы.

Для эффективной и безопстной работы с SMC необходимо разработать специализированный язык программирования и среду разработки для работы с этим языком соответсвенно. Тогда и только тогда появится возможность опираться на SMC не только как на особый «прием», а как на самодостаточный подход к проектированию систем.

Подводя итоги закреплю за этой концепцией два понятия:

- Code-oriented design (COD) – подход, при котором машинный код воспринимается и используется как данные;
- Code-oriented programming (COP) – методология программирования, основанная на видении кода программы как набора начальных данных, осуществление изменения которых в режиме реального времени позволяет достигнуть поставленной задачи.

Использование

1. Скэнлон Лео Джон. Персональные ЭВМ IBM PC и XT. / Программирование на языке ассемблера – Москва: Издательство «Радио и связь». Редакция переводной литературы, 1989.
2. Э. Танненбаум. Архитектура компьютера / Structured Computer Organization. — 5-е изд. — СПб.: Питер, 2013. — С. 476. — 884 с. — ISBN 978-5-469-01274-0

Панкевич Даниил Сергеевич, студент специальности информационных систем и технологий в игровой индустрии БГУИР, daniil.pankevich@gmail.com.

Научный руководитель: Рак Татьяна Александровна, старший преподаватель кафедры вычислительных методов и программирования БГУИР, tatianarak@bsuir.by.