Секция 7

МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ

УДК 004.431, 004.4'42, 004.451 DOI 10.17223/2226308X/8/49

МОДИФИКАЦИЯ ЛЯПАСА ДЛЯ РАЗРАБОТКИ ОС

С. Ю. Гречнев, Д. А. Стефанцов

Описывается модификация транслятора ЛЯПАС для возможности разработки операционной системы. Изменены инициализирующий и завершающий ассемблерные коды, скорректирована работа с комплексами, оптимизировано использование памяти программой, добавлены новые возможности: доступ к произвольной ячейке оперативной памяти и получение адреса процедур.

Ключевые слова: *ЛЯПАС, операционная система, Русский язык программирования.*

Разработка операционной системы (ОС) предъявляет к языку программирования и компилятору с него ряд специфических требований, например необходимость хранения служебных структур данных на протяжении всего времени работы ОС, возможность записи адресов подпрограмм в переменные языка, возможность генерирования ассемблерного кода без привязки к существующей ОС, в которой ведётся разработка, и к её формату исполняемого файла. В данной работе сообщается о модификации Логического Языка для Представления Алгоритмов Синтеза (ЛЯПАСа) [1] и соответствующего транслятора для разработки ОС [2]. Кроме упомянутых изменений, в язык добавлены возможности, повышающие удобство разработки программ на нём, а в компилятор — повышающие надёжность генерируемой ассемблерной программы.

Для работы операционной системы необходимо хранить некоторые структуры данных от запуска компьютера до его выключения. Примерами таких структур являются глобальная таблица дескрипторов (GDT), таблица дескрипторов прерываний (IDT) [3], таблица планировщика процессов. Однако в ЛЯПАСе отсутствуют глобальные переменные, обычно применяемые для данных целей, поэтому в язык добавлен доступ к произвольной ячейке оперативной памяти. Такой доступ осуществляется через специальный комплекс K, i-й элемент которого представляет собой ячейку оперативной памяти по адресу i. Дополнительно это позволяет выводить сообщения на экран, производя их запись в участок памяти с фиксированным адресом — видеобуфер.

Для построения IDT введена возможность получения адресов подпрограмм. Во внутреннюю переменную τ помещается адрес ассемблерной метки, с которой начинается подпрограмма.

Перечисленные возможности считаются потенциально небезопасными для разработки пользовательских программ и включаются при трансляции флагом `-os`.

Для компиляции файла, запускаемого в ОС GNU/Linux, транслятор добавляет инициализирующий и завершающий коды. Инициализирующий код запрашивает у ОС память под комплексы в куче, переходит на первую подпрограмму, после завершения выполняет системный вызов `sys_exit`; завершающий код содержит подпрограмму

`_addmem`, которая запрашивает дополнительную память в куче. Изменение инициализирующего и завершающего кодов включается флагом `-os`.

Дополнительно к описанным в ЛЯПАС внесены изменения, делающие язык более удобным. Во-первых, в работе с комплексами добавлена проверка по ёмкости. При обращении к i-му элементу комплекса ёмкостью S проверяется условие $i < S$. Если оно не выполняется, программа переходит на метку `_errend`.

Во-вторых, оптимизировано использование памяти программой. Ранее в ЛЯПАСе для каждой вызываемой подпрограммы в стеке выделялось 1420 байт под все возможные локальные переменные, параметры всех возможных комплексов (тип комплекса, адрес комплекса в куче, ёмкость, мощность), адрес начала свободной памяти в куче. Адреса локальных переменных фиксировались в стеке в соответствии с их именами: вызывающая подпрограмма выделяла память в стеке для вызываемой, помещала туда входные параметры, копировала значения выходных параметров после завершения подпрограммы. Данный способ прост в реализации, однако неэффективно использует память и создаёт много вспомогательных конструкций в коде.

Предлагаются следующие изменения. Память выделяется только под используемые локальные переменные; переменные и комплексы размещаются в стеке в порядке их упоминания, что позволяет намного эффективнее использовать память. Вызывающая подпрограмма выделяет память в стеке для входных и выходных параметров вызываемой подпрограммы, заполняет входные параметры и отдаёт управление вызываемой. Последняя самостоятельно выделяет память в стеке под локальные переменные, а после завершения работы удаляет их. После этого вызывающая подпрограмма копирует значения выходных параметров в свои локальные переменные и возвращает стек к исходному состоянию.

Новая организация памяти программы позволяет также реализовать эффективную композицию подпрограмм. Обозначим множество значений переменных V, множества значений символьных и логических комплексов F и L соответственно. Тогда каждая подпрограмма соответствует функции из декартова произведения, составленного из V, F, L, в другое декартово произведение, составленное аналогичным образом. Например, подпрограмма с заголовком $f(c, a, \mathrm{L1}, \mathrm{F3}/b, \mathrm{L2})$ соответствует функции $f : V \times V \times L \times F \to V \times L$. Будем говорить, что существует композиция подпрограмм f и g, если существует композиция соответствующих функций. Значения входных и выходных параметров подпрограммы будем называть входными и выходными значениями подпрограммы соответственно.

Ранее в языке для получения значения $(gf)(a)$ композиции подпрограмм f и g необходимо было вызвать подпрограмму f на a, скопировать её выходные значения в локальные переменные, подать их на вход функции g и получить её выходные значения. При новой организации памяти можно реализовать более эффективную композицию подпрограмм. Для этого после завершения работы одной подпрограммы стек не возвращается к исходному состоянию, выходные значения завершённой подпрограммы становятся входными значениями следующей в композиции подпрограммы, в стеке выделяется память под выходные значения последней, после чего происходит её вызов. Это позволяет избежать лишнего копирования данных и использования локальных переменных для передачи значений с выхода одной подпрограммы на вход другой.

В ЛЯПАСе вызов функции обозначается следующим образом: $*f(in/out)$, где f — имя подпрограммы; in — список входных значений; out — список выходных значений. Для удобства чтения и понимания программ на ЛЯПАСе композиция обозначена схожим образом: $*f_n*f_{n-1}\ldots*f_1(in_1/out_n)$, где f_1, …, f_n — названия подпрограмм;

in_1 — входные значения первой подпрограммы; out_n — выходные значения последней. При этом компилятор проверяет существование композиции подпрограмм. Для увеличения числа возможных композиций подпрограмм предлагается зафиксировать порядок перечисления входных и выходных параметров: сначала описываются параметры-переменные, затем — параметры-логические комплексы, затем — параметры-символьные комплексы. Эта особенность языка будет полезна при создании крупных библиотек на ЛЯПАСе.

ЛИТЕРАТУРА

1. *Агибалов Г. П., Липский В. Б., Панкратова И. А.* О криптографическом расширении и его реализации для русского языка программирования // Прикладная дискретная математика. 2013. № 3. С. 93–104.
2. *Стефанцов Д. А., Томских П. А.* Разработка операционной системы на языке ЛЯПАС // Прикладная дискретная математика. Приложение. 2015. № 8. С. 134–135.
3. Intel 64 and IA-32 Architectures Software Developer Manuals `http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html`

УДК 519.681.2 DOI 10.17223/2226308X/8/50

ОПЕРАЦИОННАЯ СЕМАНТИКА ЛЯПАСА

А. О. Жуковская, Д. А. Стефанцов

Сообщается о разработке операционной семантики ЛЯПАСа. Описываются её возможные применения: доказательство методом абстрактной интерпретации корректности обращения к элементам комплекса и создание верифицирующего транслятора.

Ключевые слова: *операционная семантика, ЛЯПАС, абстрактная интерпретация, верифицирующий транслятор.*

Язык программирования ЛЯПАС разработан в 1960-х годах в Томском государственном университете, использовался в СССР и других странах, в настоящее время возрождается на кафедре защиты информации и криптографии ТГУ с целью разработки доверенного программного обеспечения [1]. В данной работе сообщается об операционной семантике ЛЯПАСа, которая может применяться как минимум для двух целей: для доказательства формальных свойств программ на ЛЯПАСе и для создания верифицирующего транслятора.

Семантика языка программирования — формализация значений конструкций языка построением их математических моделей. Существует несколько вариантов семантик, самые распространённые — операционные и денотационные семантики. Денотационные семантики основаны на абстракции функций и ориентированы на функциональные языки. Операционные семантики основаны на определении состояний абстрактной машины и переходов между ними. Семантические правила представляют собой переход из одного состояния в другое при соблюдении условий. Для ЛЯПАСа и метода абстрактной интерпретации наиболее подходящий вариантом является операционная семантика.

Под программой понимается синтаксически правильная последовательность команд, представленная нумерованным списком. Состоянием программы на ЛЯПАСе является шестёрка объектов (c, var, τ, EL, Q, S), где c — номер текущей команды; var — массив значений переменных; τ — значение внутренней переменной; EL — массив спис-