БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Кафедра информатики

Факультет: ИНО
Специальность: ИиТП

Курсовая работа
по дисциплине “Операционные системы и среды”
“Виртуальная машина для интерпретатора языка программирования Python 3.8”

Выполнил студент: Дубейковский А.А.
Группа № 893551
Зачётная книжка № 75350046

Содержание

Виртуальные машины и их типы

Виртуальная машина (ВМ) — это виртуализация или эмуляция компьютерной системы. Виртуальные машины основаны на компьютерных архитектурах и обеспечивают функциональность физического компьютера. Их реализации могут включать специализированное оборудование, программное обеспечение или их комбинацию.

Виртуальные машины различаются и организованы по следующим функциям:

1. Системные виртуальные машины (также называемые виртуальными машинами полной виртуализации) заменяют реальную машину. Они предоставляют функциональные возможности, необходимые для работы целых операционных систем. Гипервизор использует родной для машины подход (машинный код) совместного использования оборудования и управления им, что позволяет использовать несколько сред, которые изолированы друг от друга, но существуют на одной физической машине. Современные гипервизоры используют аппаратную виртуализацию с аппаратной поддержкой, специфичное для виртуализации оборудование, в основном центральные процессоры.

2. Виртуальные машины процессов предназначены для выполнения компьютерных программ в независимой от платформы среде.

Некоторые эмуляторы виртуальных машин, такие как QEMU и эмуляторы игровых консолей, также предназначены для имитации различных системных архитектур, что позволяет выполнять программные приложения и операционные системы, написанные для другого процессора или другой архитектуры. Виртуализация на уровне операционной системы позволяет разделять ресурсы компьютера через ядро.

Системные виртуальные машины

“Виртуальная машина” была первоначально определена Попеком и Голдбергом как “эффективная изолированная копия реальной компьютерной машины”. Текущее использование включает виртуальные машины, которые не имеют прямого соответствия какому-либо реальному оборудованию. Физическое “реальное” оборудование, на котором работает виртуальная машина, обычно называется “хостом”, а виртуальная машина, эмулируемая на этой машине, обычно называется “гостевой”. Хост может эмулировать несколько гостевых систем, каждый из которых может эмулировать разные операционные системы и аппаратные платформы.

Желание запускать несколько операционных систем было первоначальным мотивом для виртуальных машин, чтобы обеспечить разделение времени между несколькими однозадачными операционными системами. В некоторых отношениях системную виртуальную машину можно рассматривать как обобщение концепции виртуальной памяти, которая исторически предшествовала ей. IBM CMS, первая система, допускающая полную виртуализацию, реализовала разделение времени, предоставив каждому пользователю однопользовательскую операционную систему, Conversational Monitor System (CMS). В отличие от виртуальной памяти, системная виртуальная машина дает пользователю право писать привилегированные инструкции в своем коде. Этот подход имел определенные преимущества, такие как добавление устройств ввода / вывода, не разрешенных стандартной системой.

Поскольку технология использует виртуальную память для целей виртуализации, новые системы избыточного использования памяти могут применяться для управления совместным использованием памяти несколькими виртуальными машинами в одной компьютерной операционной системе. Страницы памяти, которые имеют идентичное содержимое, могут быть совместно использованы несколькими виртуальными машинами, работающими на одной физической машине, что может привести к их сопоставлению с одной и той же физической страницей с помощью метода, называемого объединением одной страницы ядра. Это особенно полезно для страниц, предназначенных только для чтения, таких как страницы, содержащие сегменты кода, что имеет место для нескольких виртуальных машин с одинаковым или похожим программным обеспечением, программными библиотеками, веб-серверами, компонентами промежуточного программного обеспечения и т.д. Гостевым операционным системам не требуется совместимость с аппаратным обеспечением хоста, что позволяет запускать разные операционные системы на одном компьютере (например, Windows, Linux или предыдущие версии операционной системы) для поддержки будущего программного обеспечения.

Использование виртуальных машин для поддержки отдельных гостевых операционных систем очень популярно. Типичное использование - запуск операционной системы в реальном времени одновременно с предпочтительной сложной операционной системой, такой как Linux или Windows. Еще одно применение - новое и непроверенное программное обеспечение, все еще находящееся на стадии разработки, запускается внутри песочницы. Виртуальные машины имеют и другие преимущества для разработки операционных систем, а также могут включать улучшенный доступ для отладки и более быструю перезагрузку.

Виртуальные машины процессов

Виртуальная машина процесса, иногда называемая виртуальной машиной приложения или управляемой средой выполнения (Managed Runtime Environment), работает как обычное приложение внутри ОС хоста и поддерживает один процесс. Он создается при запуске этого процесса и уничтожается при выходе. Его цель - предоставить платформенно-независимую среду программирования, которая абстрагирует детали базового оборудования или операционной системы и позволяет программе выполняться одинаково на любой платформе.

Виртуальная машина процесса обеспечивает абстракцию высокого уровня - абстракцию языка программирования высокого уровня (по сравнению с абстракцией ISA низкого уровня системной виртуальной машины). Виртуальные машины процессов реализуются с помощью интерпретатора; производительность, сравнимая с компилируемыми языками программирования, может быть достигнута за счет использования ‘just-in-time' компиляции.

Этот тип ВМ стал популярным в языке программирования Java, который реализован с помощью виртуальной машины Java. Другие примеры включают виртуальную машину Parrot и .NET Framework, работающую на виртуальной машине, которая называется Common Language Runtime. Все они могут служить слоем абстракции для любого компьютерного языка. Именно такую виртуальную машину я сделал для Python 3.8, подробности про которую будут дальше.

Частным случаем процессных виртуальных машин являются системы, которые абстрагируются от механизмов связи компьютерного кластера. Такая виртуальная машина состоит не из одного процесса, а из одного процесса на физическую машину в кластере. Они предназначены для облегчения задачи программирования параллельных приложений, позволяя программисту сосредоточиться на алгоритмах, а не на механизмах связи, обеспечиваемых соединениями машин и ОС. Они не скрывают того факта, что происходит коммуникация, и не пытаются представить кластер как единую машину.

В отличие от других виртуальных машин процесса, эти системы не предоставляют определенный язык программирования, а встроены в существующий язык; обычно такая система обеспечивает привязки для нескольких языков (например, C и Fortran). Примерами являются параллельная виртуальная машина (Parallel Virtual Machine) и интерфейс передачи сообщений (Message Passing Interface). Они не являются строго виртуальными машинами, потому что приложения, работающие над ними,

по-прежнему имеют доступ ко всем службам ОС и, следовательно, не ограничиваются системной моделью.

Язык программирования Python - краткое введение

Python — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным в том плане, что всё является объектами. Необычной особенностью языка является выделение блоков кода пробельными отступами. Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации. Сам же язык известен как интерпретируемый и используется в том числе для написания скриптов. Недостатками языка являются зачастую более низкая скорость работы и более высокое потребление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на компилируемых языках, таких как Си или C++.

Python является мультипарадигмальным языком программирования, поддерживающим императивное, процедурное, структурное, объектно-ориентированное программирование, метапрограммирование и функциональное программирование. Задачи обобщённого программирования решаются за счёт динамической типизации. Аспектно-ориентированное программирование частично поддерживается через декораторы, более полноценная поддержка обеспечивается дополнительными фреймворками. Такие методики как контрактное и логическое программирование можно реализовать с помощью библиотек или расширений. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений с глобальной блокировкой интерпретатора (GIL), высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Эталонной реализацией Python является интерпретатор CPython, поддерживающий большинство активно используемых платформ и являющийся стандартом де-факто языка. Он распространяется под свободной лицензией Python Software Foundation License, позволяющей использовать его без ограничений в любых приложениях, включая проприетарные. CPython компилирует исходные тексты в высокоуровневый байт-код, который исполняется в стековой виртуальной машине. К другим трём основным реализациям языка относятся Jython (для JVM), IronPython (для CLR/.NET) и PyPy. PyPy написан на подмножестве языка Python (RPython) и

разрабатывался как альтернатива CPython с целью повышения скорости исполнения программ, в том числе за счёт использования JIT-компиляции. Поддержка версии Python 2 закончилась в 2020 году. На текущий момент активно развивается версия языка Python 3. Разработка языка ведётся через предложения по расширению языка PEP (Python Enhancement Proposal), в которых описываются нововведения, делаются корректировки согласно обратной связи от сообщества и документируются итоговые решения.

Стандартная библиотека включает большой набор полезных переносимых функций, начиная от функционала для работы с текстом и заканчивая средствами для написания сетевых приложений. Дополнительные возможности, такие как математическое моделирование, работа с оборудованием, написание веб-приложений или разработка игр, могут реализовываться посредством обширного количества сторонних библиотек, а также интеграцией библиотек, написанных на Си или C++, при этом и сам интерпретатор Python может интегрироваться в проекты, написанные на этих языках. Существует и специализированный репозиторий программного обеспечения, написанного на Python, — PyPI. Данный репозиторий предоставляет средства для простой установки пакетов в операционную систему и стал стандартом де-факто для Python. По состоянию на 2019 год в нём содержалось более 175 тысяч пакетов.

Python стал одним из самых популярных языков, он используется в анализе данных, машинном обучении, DevOps и веб-разработке, а также в других сферах, включая разработку игр. За счёт читабельности, простого синтаксиса и отсутствия необходимости в компиляции язык хорошо подходит для обучения программированию, позволяя концентрироваться на изучении алгоритмов, концептов и парадигм. Отладка же и экспериментирование в значительной степени облегчаются тем фактом, что язык является интерпретируемым. Применяется язык многими крупными компаниями, такими как Google или Facebook. По состоянию на октябрь 2021 года Python занимает первое место в рейтинге TIOBE популярности языков программирования с показателем 11,27%. «Языком года» по версии TIOBE Python объявлялся в 2007, 2010, 2018 и 2020 годах. И на данный момент (22 Декабря 2021 года) держит первое место по популярности.

Стандартная реализация языка программирования Python - CPython

CPython - эталонная реализация языка программирования Python. Написанный на C и Python, CPython является стандартной и наиболее широко используемой реализацией языка Python.

CPython можно определить и как интерпретатор, и как компилятор, поскольку он компилирует код Python в байт-код перед его интерпретацией.

У него есть интерфейс внешней функции с несколькими языками, включая C, в котором нужно явно писать привязки на языке, отличном от Python.

Например, вызов функции print() представляется в байт-коде в следующем виде:

```
0 LOAD_NAME                0 (print)
2 LOAD_CONST               0 ('Hello World!')
4 CALL_FUNCTION            1
6 RETURN_VALUE
```

Особенностью CPython является то, что он использует глобальную блокировку интерпретатора (GIL) для каждого процесса интерпретатора CPython, что означает, что в рамках одного процесса только один поток может одновременно обрабатывать байт-код Python. Это не означает, что в многопоточности нет смысла; наиболее распространенный сценарий многопоточности — это когда потоки в основном ждут завершения внешних процессов.

Это может произойти, когда несколько потоков обслуживают отдельных клиентов. Один поток может ждать ответа от клиента, а другой может ждать выполнения запроса к базе данных, в то время как третий поток фактически обрабатывает код Python.

Однако GIL действительно означает, что CPython не подходит для процессов, реализующих алгоритмы, интенсивно использующие ЦП, в коде Python, который потенциально может быть распределен по нескольким ядрам.

В реальных приложениях ситуации, когда GIL является значительным узким местом, довольно редки. Это связано с тем, что Python по своей сути является медленным языком и обычно не используется для операций, интенсивно использующих процессор или требующих времени. Python обычно используется на верхнем уровне и вызывает функции в библиотеках для выполнения специализированных задач. Эти библиотеки обычно не написаны на Python, и код Python в другом потоке может выполняться во время вызова одного из этих базовых процессов. Библиотека, отличная от Python, вызываемая для выполнения задачи, требующей интенсивного использования ЦП, не подчиняется GIL и может одновременно выполнять множество потоков на нескольких процессорах без ограничений.

Параллелизм кода Python может быть достигнут только с помощью отдельных процессов интерпретатора CPython, управляемых многозадачной операционной системой. Это усложняет обмен данными между параллельными процессами Python, хотя модуль многопроцессорности

несколько смягчает это; это означает, что приложения, которые действительно могут извлечь выгоду из параллельного выполнения кода Python, могут быть реализованы с ограниченными накладными расходами.

Наличие GIL упрощает реализацию CPython и упрощает реализацию многопоточных приложений, которые не выигрывают от параллельного выполнения кода Python. Однако без GIL многопроцессорные приложения должны гарантировать, что весь общий код является потокобезопасным.

Хотя было сделано много предложений по устранению GIL, по общему мнению, в большинстве случаев преимущества GIL перевешивают недостатки; в тех немногих случаях, когда GIL является узким местом, приложение следует строить на основе многопроцессорной структуры.

Моя виртуальная машина для интерпретатора Python 3.8

Виртуальная машина создавалась для интерпретатора последней версии Python к моменту её создания - Python 3.8. Реализованного через CPython. Посмотрим на то, как происходит исполнение кода Python по шагам своим родным интерпретатором:

1. У нас есть какой-то объект с кодом (.py файл или несколько файлов с кодом)
2. Мы запускаем его с помощью интерпретатора Python, который в первую очередь токенезирует весь код (в Python есть библиотека ‘tokenize’ написанная на C, которая может помочь это сделать). Токенизация – это процесс получения текстового потока данных и разбиения на токены значащих (для интерпретатора) слов с дополнительными метаданными (например, где токен начинается и кончается, и каково строковое значение этого токена).
3. Дальше все токены парсятся использую грамматические и синтаксические особенности языка Python (библиотека ‘parser’ может упростить выполнение этой задачи), чтобы построить абстрактное синтаксическое дерево. В виде такого дерева любой код представляется для машины.
4. Теперь машина может исполнить этот код пользуясь абстрактным синтаксическим деревом. Например, для языков C, C++, Go и многих других синтаксическое дерево будет скомпилировано в машинный код. В случае CPython реализации языка Python (далее будем считать, что весь наш Python код используется стандартную реализацию через CPython), абстрактное синтаксическое дерево будет переведено в байт-код (пример которого был в предыдущей главе про CPython). Библиотеки ‘ast’ или ‘byteplay’ от Google могут помочь решить эту задачу.

5. Теперь нужно исполнить байт-код. В отличии от машинного кода, который исполняется напрямую машиной, байт-код исполняется виртуальной машиной. Именно такую виртуальную машину мы и сделаем для байт-кода.

Уточню, что создание этой виртуальной машины было одним из главных семестровых домашних заданий по курсу Advanced Python в Школе Анализа Данных от Яндекса. Её готовая реализация с кодом, подробными комментариями по реализации кода и пояснениями как её использовать выложена на моём гит-хаб аккаунте: https://github.com/ElephantT/A-Python-Interpreter-Written-in-Python

Виртуальная машина будет написана с помощью языка программирования Python, но у нас будут ограничения на прямое использование родного интерпретатора Python, так как мы хотим сделать его полностью сами. Ограничить использование родного интерпретатора можно за счёт установления невозможности использования следующих возможностей языка Python:

1. Функции exec, eval, inspect
2. Тип FunctionType
3. И подобный функционал Python, где code — это наш получаемый байт-код:
   a.
```
def f():
    pass
f.__code__ = code
f()
```

Я бы не буду переносить весь код сюда, т.к. он достаточно объёмный, но не сложный для понимания (сам код для виртуальной машины - 1000 строк, код для прогона тестов - 500 строк, и тесты - 3500 строк), поэтому все его главные части я объясню ссылаясь на местоположения кода на гит-хабе, и копируя лишь какие-то главные объекты. Напомню, что в коде есть качественные комментарии к каждому его ключевому элементу:

1. Файл ‘vm_runner.py’:
   a. Содержит функцию ‘compile_code’, которая получает на вход либо уже скомпилированный код, либо обычный текст с кодом Python, который затем компилируется в байт-код (токенизируется, парсится, строится абстрактное синтаксическое дерево и компилируется в байт-код). Для этого используется библиотека dis, которая сильно упрощает выполнение всех этих шагов.
   b. Остальные функции в этом файле могут исполнить код, посмотреть его выводы, исключения и ошибки, и сохранить их, используя родной интерпретатор Python и его родную

виртуальную машину. Это поможет нам тестировать нашу виртуальную машину.

2. Файл ‘function_type_ban.py’:
    a. Задача объектов этого класса - проверка, использовался ли родной интерпретатор Python для реализации нашей виртуальной машины. Подробнее про то, как можно ограничить использование нативного интерпретатора написано выше.
3. Файл ‘cases.py’:
    a. Тут содержится около 320 тестов, каждый из которых содержит какой-то код написанный на Python. Этот код представляется как обычная строка символов.
    b. Данные тесты покрывают все возможности языка Python 3.8. То есть прохождение всех тестов будет означать, что наша виртуальная машина может исполнять любой код, написанный на Python 3.8.
4. Файлы ‘test_public.py’, ‘test_stat.py’ и ‘vm_scorer.py’:
    a. Используются для прогона тестов.
    b. ‘vm_scorer.py’ содержит информацию об оценках в баллах каждого теста (использовались, чтобы оценить выполнение данной семестровой задачи). А также возможность посмотреть сколько тестов пройдено из каждого разбиения тестов. Примеры разбиений:
        i. 1) тесты, покрывающие всевозможное использование унарных операторов в коде
        ii. 2) тесты, покрывающие всевозможное использование бинарных операторов в коде
        iii. 3) тесты, покрывающие всевозможное использование выбрасывание исключений в коде
        iv. 4) и т.д.
    c. Другие два файла позволяют запустить все тесты, вывести в консоль результаты, например посмотреть сколько из тестов мы прошли, какие темы насколько мы покрыли, сколько баллов получили.
5. Файл ‘vm.py’:
    a. Это сама виртуальная машина.
    b.
```
783  class VirtualMachine:
784      def run(self, code_obj: types.CodeType) -> None:
785          """
786          :param code_text_or_obj: code for interpreting
787          """
788          globals_context: tp.Dict[str, tp.Any] = {}
789          frame = Frame(code_obj, builtins.globals()['__builtins__'], globals_context, globals_context)
790          return frame.run()
```
    c. Frame (реализация: 162 - 781 строки этого файла):
        i. Следит за частью исполняемого на данный момент кода, а именно за тем, что лежит в стэке, какие у нас есть глобальные переменные, какие у нас есть локальные

переменные, различные виды аргументов, которые есть у данного фрейма, хранит исполняемый байт-код, хранит последнее выброшенное исключение, хранит результат исполнения фрейма и т.д.

ii. Также он содержит реализации всех возможных операций байт-кода, а именно, как их нужно исполнить нашей виртуальной машине (нашему интерпретатору).

iii. По своей сути, сколько именно операций байт-кода мы сможем реализовать, такое покрытие кода Python наша виртуальная машина и будет покрывать (сможет интерпретировать).

Моя реализация виртуальной машины покрывает 220 тестов. Они составляют покрытие всех частей Python 3.8, за исключением поддержки классов и импортирования других модулей, которые и включают большинство из оставшихся 96 не пройденных тестов. Более подробную информацию о непокрытых тестах можно найти в файле ‘vm_results.txt’ сверяя с тестами из ‘cases.py’.

Источники

1. "Formal requirements for virtualizable third generation architectures" Popek, Gerald J.; Goldberg, Robert P. (1974)
2. "The Architecture of Virtual Machines" Smith, James E.; Nair, Ravi (2005)
3. https://www.tiobe.com/tiobe-index/
4. https://azure.microsoft.com/ru-ru/overview/what-is-a-virtual-machine
5. Подобный академический проект для Python 2.7 и Python 3.3 https://github.com/nedbat/byterun
6. Мой интерпретатор для Python 3.8 CPython https://github.com/ElephantT/A-Python-Interpreter-Written-in-Python
7. Реализация родного интерпретатора Python https://github.com/python/cpython/blob/3.8/Python/ceval.c