**2. Основные компоненты MRE и их назначение**



Инкапсуляция — принцип модульности, система разделяется на модули, «внутренности» которых скрыты от остальных.

Среды управляемого исполнения — экзоскелет вокруг пользовательского кода.

Цена модульности — накладные расходы при переходе от одного модуля к другому.

**Менеджер памяти** — выделяет память, обнаруживает живые объекты, собирает мусор (освобождает память).

Для каждого семейства сборщиков мусора — своя стратегия выделения памяти. (GC = Garbage Collector = сборщик мусора)

**Планировщик** — следит за компонентами системы и синхронизацией (основной вид синхронизации — исполнение большого числа программ на малом числе процессоров)

* Создание и разрушение нитей
* Распределение нагрузки между процессорами
* Приостановка и возобновление нитей
* Для синхронизации управляемого кода
* Для сборки мусора
* Для синхронизации неуправляемого кода и сборки мусора

**Исполнитель**

*Профилятор* — постоянно профилирует (oh, really? :D). С помощью профилятора достигается локальный оптимальный баланс между скоростью работы и временем компиляции.

(Профилирование — сбор характеристик работы программы, таких как время выполнения отдельных фрагментов (обычно подпрограмм), число верно предсказанных условных переходов, число кэш-промахов и т. д.)

Интерпретатор — медленный, работает через внешний интерфейс.

(Интерпретация — пооператорный (покомандный, построчный) анализ, обработка и тут же выполнение исходной программы или запроса.)

JIT-компиляция (Just-in-time compilation) —компиляция байт-кода в машинный код или в другой формат непосредственно во время работы программы.

Процесс компиляции состоит из следующих этапов:

* Лексический анализ. На этом этапе последовательность символов исходного файла преобразуется в последовательность лексем.
* Синтаксический (грамматический) анализ. Последовательность лексем преобразуется в дерево разбора.
* Семантический анализ. Дерево разбора обрабатывается с целью установления его семантики (смысла) — например, привязка идентификаторов к их декларациям, типам, проверка совместимости, определение типов выражений и т. д. Результат обычно называется «промежуточным представлением/кодом», и может быть дополненным деревом разбора, новым деревом, абстрактным набором команд или чем-то ещё, удобным для дальнейшей обработки.
* Оптимизация. Выполняется удаление излишних конструкций и упрощение кода с сохранением его смысла. Оптимизация может быть на разных уровнях и этапах — например, над промежуточным кодом или над конечным машинным кодом.
* Генерация кода. Из промежуточного представления порождается код на целевом языке.

загрузка/cохранение Данных и кода – вроде понятно

**Загрузчик IR**

Есть возможность сохранить промежуточное представление (Intermediate Representation) (часть работы JIT — компилятора). Это ускорит последующие компиляции, однако создает уязвимости.

Один из методов борьбы с такими уязвимостями Trusted computing — все промежуточные данные подписываются цифровой подписью

* Задача IR
  + Компактность
  + Стандартизация
  + Простота интерпретации и компиляции
* Основные виды IR
  + Язык стековой машины
  + Язык регистровой машины
* Язык регистровой машины
  + На 45% меньше операций доступа к памяти
  + Возможность оптимизации на уровне IR
  + Используется в статических и оптимизирующих JIT компиляторах
* Язык стековой машины
  + На 25% более компактен
  + Сохраняет семантику исходного кода
  + Проще для интерпретации и шаблонной компиляции
  + Используется в JBC и CIL

Типы команд

* Простые
  + Стековая арифметика
  + Доступ к локальным данным
  + Ветвления
* Объектные
  + Создание объектов
  + Доступ к полям
  + Вызов методов
  + Исключения

Верификация

1. Проверка формата класса
2. Проверка данных класса
   * final классы и методы не перекрыты (От финального класса запрещено наследоваться)
   * Определен суперкласс
   * Формат пула констант
3. Проверка кода класса
   * Размер стека, тип данных на стеке
   * Правильный тип локальной переменной при доступе
   * Число аргументов при вызове метода
   * Необходимое количество и тип аргументов инструкций
4. Проверка ссылок
   * Наличие классов, методов и полей
   * Права доступа

Опр: Указатель — число, адрес ячейки памяти.

Опр: Ссылка — уникальный идентификатор какой- либо сущности.

**Ядро**

Модели данных: модели потоков, нитей, рефлексий, итд

Рефлексия (от позднелат. reflexio - обращение назад) - это механизм исследования данных о программе во время её выполнения. Рефлексия позволяет исследовать информацию о полях, методах и конструкторах классов. Можно также выполнять операции над полями и методами которые исследуются. Рефлексия в Java осуществляется с помощью Java Reflection API. Этот интерфейс API состоит из классов пакетов java.lang и java.lang.reflect. С помощью интерфейса Java Reflection API можно делать следующее:

* Определить класс объекта.
* Получить информацию о модификаторах класса, полях, методах, конструкторах и суперклассах.
* Выяснить, какие константы и методы принадлежат интерфейсу.
* Создать экземпляр класса, имя которого неизвестно до момента выполнения программы.
* Получить и установить значение свойства объекта.
* Вызвать метод объекта.
* Создать новый массив, размер и тип компонентов которого неизвестны до момента выполнения программ.

Внешний интерфейс — позволяет написать методы, для тех случаев, когда невозможно целиком написать приложение на языке Java.

Например, стандартная библиотека класса Java не поддерживает платформенные особенности или программную библиотеку.

Данный фреймворк позволяет нативному методу использовать Java объекты также, как Java код. Нативный метод может создавать Java объекты, а затем просматривать и использовать для выполнения своих задач. Нативный метод также может просматривать и использовать объекты, созданные кодом приложения Java. (непонятно)

* Вызов внешних неуправляемых функций
* Обратный вызов управляемого кода (callback) — передача внешней библиотеке указателя на функцию, которая будет обрабатывать события.
  + Два способа итераций:
    - внешняя итерация

for(i=0; i<=n; i++)  
a.next[i]

* + - внутренняя  
      a.iterate (это и есть callback)
* Передача данных между управляемым и неуправляемым кодом (marshaling)
  + Копирование — эффективно для малых данных и неэффективно для больших
  + Обертывание и обратный вызов — решает безопасным способом абсолютно все задачи, но есть проблемы с эффективностью
  + Закрепление объектов (pinning) — запрет объектам перемещаться из памяти
  + Указатель на неуправляемую память

Внешний интерфейс Java (Java Native Interface)

Инициализация - ?