***15. Метрики скорости системы управления памятью. Способы выделения памяти.***

*Менеджер памяти*

* Задачи
  + Выделение памяти под объекты
  + Обнаружение живых объектов
  + Сборка мусора
* Свойства приложений
  + Структура выделения памяти (allocation pattern)
    - количество и размер выделяемых объектов
  + Коэффициент смертности (mortality ratio)
    - процент объектов, не переживших сборку мусора

*Метрики Скорости*

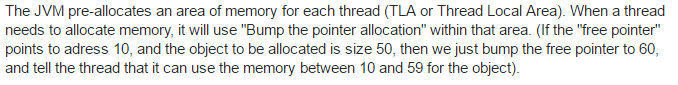
* Производительность выделения памяти
  + Объектов в сек, Кб в секунду
* Задержка выделения памяти
  + Среднее и максимальное время выделения объекта
* Производительность сборки мусора
  + Объектов в сек, Кб в секунду
* Задержка сборки мусора
  + Среднее и максимальное время выделения объекта приостановки нити для сборки мусора

*Способы достижения скорости*

* Уменьшение сложности
  + 2 шаговые алгоритмы – быстрый частый шаг (fast path), медленный редкий шаг (slow path)
* Распараллеливание операций с памятью (parallel GC, Thread local allocation)
  + Уменьшение синхронизации
  + Использование локальных данных нити
  + Вынесение синхронизации в медленный шаг
* Разбиение на порции (incremental GC)
* Параллельное исполнение с основным кодом (concurrent GC)

*Выделение памяти*

* Двухшаговое выделение памяти
  + Медленный шаг. Выделение буфера памяти для нити из глобальной кучи с синхронизацией
  + Быстрый шаг. Выделение памяти под объект из буфера нити без синхронизации
* Список свободных блоков (free list)
  + Список или дерево блоков различного размера. При выделении находится первый подходящий блок. Остаток вставляется в список.
  + Раздельный список свободных блоков по размерам, обычно 2n (segregated free list). Остаток блока разбивается на куски нужного размера и вставляется в список.
* Последовательное выделение (bump pointer)



* + Вершина кучи сдвигается на размер нового объекта

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Это надо?  
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Походу нет, то, что ниже - это уже сборка мусора**

*Перечисление живых объектов: Сканирование кучи*

* Алгоритм
  1. Остановка нитей
  2. Перечисление корневых ссылок
  3. Трассировка ссылок начиная с корневых для построения транзитивного замыкания
* Достоинства
  + Прост в реализации
  + Не зависит от других подсистем
  + Не влияет на производительность управляемого кода
* Недостатки
  + Требует остановки нитей (Stop the World)
  + Не может быть разбит на порции (All or nothing)

*Перечисление корневых ссылок*

* С помощью карт ссылок (GC Maps)
  + Требует генерации карт ссылок
  + Обеспечивает точное перечисление живых объектов (strict GC)
  + Позволяет использовать поле в заголовке объекта при трассировке ссылок
* Консервативное сканирование стека (conservative stack scanning)

[conservative-garbage-collector](http://stackoverflow.com/questions/7629446/conservative-garbage-collector)

* + Не требует поддержки других подсистем
  + Не различает ссылку на объект от примитивного типа с тем же численным значением. Ссылки на необъекты.
  + Не позволяет использовать поле в заголовке объекта при трассировке ссылок из-за ссылок на необъекты.

*Параллельная трассировка ссылок*

* Один бит в заголовке объекта
* Алгоритм
  1. В начале все объекты белые
  2. Каждая нить начинает со своего множества корневых ссылок
  3. Посещенные объекты красятся черным
  4. Отслеживаются ссылки на белые объекты (поиск в ширину или глубину)
  5. Алгоритм завершается, когда не осталось ссылок на белые объекты у всех нитей
* Оставшиеся в куче белые объекты мертвы
* Не требуется синхронизация поскольку результат инвариантен относительно нити
* Фоновая синхронизация кэша процессоров для оптимизации