Introspection & Reflection

Jan Vraný

Department of Computer Science and Engineering
Czech Technical University In Prague
Faculty of Electrical Engineering

November 25, 2008

Outline

Problem

Physical memory is limited.

Thus it must be reclaimed.

How to find out which memory is no longer used and can be (safely) reclaimed?

Terminology

mutator & collector

root-set

conservatism

Outline

Reference Counting

Principle

A reference counter is associated with each object

When this counters decreases to zero, object is destroyed

Reference Counting

Principle

A reference counter is associated with each object

When this counters decreases to zero, object is destroyed

Mark-and-sweep

Algorithm

1. All objects are colored **white**
2. Marking phase: traverse all objects (starting with root set), mark all reached objects **black**
3. Sweep phase: all **white** objects are destroyed

Copy Collector

- Separates memory into two spaces:

  from-space
  to-space

Algorithm

1. Traverse all objects (starting with root-set), copy each reached object into the *to-space*
2. *to-space* becomes *from-space* and vice versa

Outline

Motivation

- Pausing whole application because of garbage collection is simply unacceptable
- Incremental approaches allow mutator and collector to be interleaved or run in parallel.

Coordination of mutator & collector

Tricolor marking

- Objects that have been traversed are marked **black**
- Objects that have been traversed *but that its descendant may not have been* are marked **gray**
- Rest are marked **white**

Barriers

- Write Barrier
- Read Barrier

Write Barrier

Detects mutators' attempts to modify an object.

What makes GC life hard?

If mutator:

1. writes a pointer to a white object into the a black object
2. destroys pointer to the original object before GC sees it

Write Barrier Approaches

- snapshot-at-beggining
- incremental update

Read Barrier

When the mutator accesses an object, the object is *immediately* colored **gray**.

Baker's Incremental Copying

Any *from-space* object accesses by the mutator if first copied to the *to-space*.

Implementation of Barriers

- mutator must call the GC to perform some action
- In practice, this action is relatively simple
- Thus:
  - barriers can be easily *inlined* into the mutator's code
  - barriers can be supported/implemented directly by the hardware

Outline

Lifetime

Observation

Most objects live a very short time, while small percentage of them live much longer

Lifetime

Observation

Most objects live a very short time, while small percentage of them live much longer

Idea

Segregate object by age into multiple regions

Scan regions with older objects less often.

Outline

History Overview :-)

1978 - Baker's Incremental Copy Collector

1983 - Generational GC

1992 - Smalltalk/X 1.0 released (generational GC)

1995 - release of Java & JVM (mark-and-sweep only)

2002 - release of Java 1.4.0 (generational GC)

Summary

- three basic GC methods:
  - reference-counting
  - mark & sweep
  - copy collector
- incremental garbage collection, barriers
- generational garbage collection