Garbage Collection

marcel.hlopko@fit.cvut.cz

Problem

We need to allcoate memory, and we have to free it back.

Manual Deallocation

- × hinders design and extensibility
- × forces bookkeeping
- × GC can be faster[1], usually comparative[2]

Manual Deallocation

A lot of large applications ended up implementing some form of automatic memory management themselves.

Let's Make a GC

- How to find live / gargabe objects?
- How to reclaim garbage?

Object liveness

Root Set & Reachability

Finding Live Objects

- Reference Counting
- Tracing

Reference Counting

- **refCount** stored in object header
- object reclaimed when **refCount** reaches 0

Reference Counting

- ✓ incremental nature
- ✓ easy to make real-time
- ✓ degrades well with full heap
- × **cycles**
- × overhead
- × fragmentation

Can we do better?

Tracing algorithms

- GC is not running always
- GC scans the whole heap

Mark-Sweep

- marking live objects
- sweeping all unmarked

Mark-Sweep

- ✓ handles cycles
- × fragmentation
- × bigger heap - longer run

Mark-Compact

How about "defragmenting" living objects?

- ✓ solves fragmentation
- ✓ solves referential locality
- ✓ simple allocation
- × still multiple passes over heap

Baker

How about copying objects on the fly?

2 semispaces, only one used at the time

Baker

- ✓ only one run over the heap
- × memory demanding
- × still has to stop the world

Incremental Collectors

GC runs in parallel with application (**mutator**)

relaxed consistency

(conservative approximation of the true reachability graph)

Tricolor Marking

black will be retained
white will be collected
grey will be expanded

Incremental Algorithm

- color root set grey
- while grey set is not empty
  1. take grey object
  2. color all his white children grey
  3. color object black

Mutator mutates

Mutator can assign a **white** child into the **black** object

Solutions:

- read barrier
- write barrier

Read Barrier

Detect reads of white objects and color them grey immediately.

Read Barrier

- ✓ mutator "keeps itself" inside grey wave
- × usually too expensive

Baker's Read Barrier GC

- **atomic** flip
- mutator resumed
- fromspace access is **trapped**
- fromspace object copied first
- copied objects colored grey

Write Barrier

Detect writes into black objects.

Snapshot-at-beginning

Does not allow the pointer to be overwritten, it stores the original pointer "off to the side".
Very **conservative**

Incremental Update

When a reference to a white object is stored into the black object, the black object can be greyed (Steele) or the white object can be greyed (Dijkstra)

Object Age

80-98% short lived objects.

Many survivors will survive a lot.

Generations

Heap will be divided into multiple sections, each containing different generations of objects.

Generations

- **young** generation, GC'ed very often (JVM default size: 640kb)
- **old** generation, GC'ed less often (JVM default size: 64mb)

Remembered Sets

References from old gen. to eden are recorded.
All such old gen. objects are part of the root set for young gen.

Questions And Discussion

Literature: Uniprocessor Garbage Collection Techniques, Paul R. Wilson