Lecture 18

Advanced Garbage Collection

I. Break Up GC in Time (Incremental)
II. Break Up GC in Space (Partial)

Readings: Ch. 7.6.4 - 7.7.4
Trace-Based GC: Memory Life-Cycle

- Mutator runs
  - free → unreached

- GC Tracing
  - Repeat until unscanned = ∅
  - Scanned objects for new reachable objects
  - reached → scanned
  - unreached → found to be reached

- GC Done tracing
  - free → unreached
  - unreached → scanned
Incremental GC

- Interleaves GC with mutator action to reduce pause time

\[
\text{Ideal} = (R \cup \text{New}) - \text{Lost} \\
(R \cup \text{New}) - \text{Lost} \subseteq \text{Answer} \subseteq (R \cup \text{New})
\]
Effects of Mutation

- **Reachable set changes as mutator runs**
  - \( R \): set of reachable objects before the mutator runs
  - **Ideal**: set of reachable objects at the end of the GC cycle
  - **New**: set of newly created objects
  - **Lost**: set of objects that become unreachable in the interim
  - **Ideal** = \( (R \cup \text{New}) - \text{Lost} \)

- **Ideal**: Very expensive

- **Conservative Incremental GC**:
  - May misclassify some unreachable as reachable
  - should not include objects unreachable before GC starts
  - guarantees that garbage will be eliminated in the next round

\[ \text{Ideal} = (R \cup \text{New}) - \text{Lost} \subseteq \text{Answer} \subseteq (R \cup \text{New}) \]
Algorithm Proposal 1

- Initial condition
  - Scanned, Unscanned lists from before

- To resume GC
  - Find root sets
  - Place newly reached objects in “unscanned list”
  - Continue to trace reachability without redoing “scanned” objects

- Did we find all reachable objects?
Missed Reachable Objects

- All reaching pointers are found in "scanned objects"
- Requires the occurrence of a 3-step sequence in the mutator:

0. after a stage of GC

1. Load $p = \text{ptr}$ from B to C

2. Store $p$ in A

3. Store new pointer in B, overwriting value $p$
Solution

• Intercept p in any of the three-step sequence
• Treat pointee of p as “unscanned”

0. after a stage of GC

1. Load p = ptr from B to C
   Read barrier: remember all loads of pointers from B → C

2. Store p in A
   Write barrier: remember all stores of pointers A → C

3. Store new pointer in B, overwriting value p
   Overwrite barrier: remember all overwrites of pointer B → C
Efficiency of Different Barriers

- **Most efficient:** Write barrier
  - less instances than read barrier
  - includes less unreachable objects than over-write barriers
II. Partial GC

• Reduces pause time by collecting only objects in the target area:

  - **Algorithm**
    - New “root set”
      - original root set + pointers from Stable to Target set
    - Change program to intercept all writes to Stable set
  
  • Never misclassify reachable as unreachable
  • May misclassify unreachable as reachable
Generational GC

• Observation: objects die young
  – 80-98% die within a few million instructions or before 1 MB has been allocated

• Generational GC: collect newly allocated objects more often

• ith generation
  – new root set
    = original root set + all pointers from generations \( j \) to \( i \) \((j > i)\)

• When 1st generation fills up
  – GC copies reachable objects into 2nd generation, and so on.
Properties

• Never misclassify reachable as unreachable

• Misclassify unreachable as reachable
  – when pointers in earlier generations are overwritten
  – eventually collect all garbage as generations get larger

• Effective: time spent on objects that are mostly garbage

• GC of mature objects takes longer
  – Size of target set increases
  – Eventually a full GC is performed
Conclusions

- **Trace-based GC**: find all reachable objects, complement to get unreachable
  - 4 states: free, unreached, unscanned, scanned
  - break up reachability analysis
    - in time (incremental)
    - in space (partial: generational)