

Exzerpt zu: *Simple Generational Garbage Collection and Fast Allocation*

Preview:

- Generational Garbage Collection
- Schnelle Allokation
- Dynamische Speicherallokation
- Mark-And-Sweep
- Copying algorithm
- Page Traps

Questions:

1. Auf welchen Beobachtungen/Annahmen basiert Generational Garbage Collection?
2. Was unterscheidet Generational Garbage Collection von Mark-and-Sweep oder einem Copying Collector?
3. Warum sind Allokationen hier schneller?
4. Was sind Page Traps?

Read:

1. Beobachtungen:
 - Junge Objekte (gerade erst allokiert) verweisen oft auf alte Objekte, alte aber nur selten auf junge Objekte.
 - Wenn ein altes Objekt auf ein junges zeigt, so muss das alte Objekt nach seiner initialisierung verändert werden (ist in Sprachen mit GC eher selten).
 - Die meisten Objekte haben eine kurze Lebensdauer. Junge Objekte haben daher eine höhere Wahrscheinlichkeit bald „Abfall“ zu werden.
 - Objekte mit längerer Lebensdauer gehören oft zu Datenstrukturen, die permanent (oder zumindest nahezu) sind.
2. Ein Generational Garbage Collector ist eine Variante eines Copying Collectors. Objekte werden in verschiedenen Speicherbereichen gehalten, wobei alle Objekte in einem Bereich etwa das selbe Alter haben. Da, nach der Vorausgehenden Beobachtungen, fast keine Zeiger von einem jüngeren in einen älteren Bereich zeigen, kann der Copying Collector Algorithmus nun immer erst auf den jüngsten Bereich angewendet werden. Dabei müssen ältere Bereiche (fast) nicht berücksichtigt werden. Es gibt mehrere Verfahren, um herauszufinden, welche Verweise von älteren Objekten auf jüngere zeigen. Diese können dann mit ins Root Set aufgenommen werden.
3. Durch das zusammenkopieren der Objekte bei jeder Bereinigung ist der komplette belegte Speicher immer ein zusammenhängender Block. Es muss also keine Liste freier Bereiche verwaltet werden und eine Allokation besteht im wesentlichen nur aus der Inkrementierung eines Pointers.
4. Hat mit der Implementierung zu tun. Ist für das allgemeine Verfahren unwichtig.

Reflect:

Der erste Teil beschreibt nochmals die Funktion eines Generational Garbage Collectors. Es wird auch nochmals kurz auf die beiden grundlegenden Verfahren Mark-and-Sweep und Copying Collector eingegangen. Der zweite Teil enthält hauptsächlich sehr auf Unix bezogene

Implementierungsdetails.

Recite:

Viele Programmierumgebungen verwenden mittlerweile automatische Speicherverwaltung und Bereinigung. Mit zunehmender Speichergröße wird der Copying Collector zur effizientesten Methode.

Es gibt jedoch einige Beobachtungen, auf deren Basis man diesen Kollektortyp verbessern kann. Diese Beobachtungen sind, dass es nur wenige Zeiger von alten auf neue Objekte gibt und junge Objekte viel häufiger schnell wieder „sterben“ als ältere. Daraus lässt sich das Konzept ableiten, den Heap in „Generationen“ einzuteilen. Nun müssen nicht jedes mal, wenn der Kollektor den Speicher bereinigt, alle Objekte überprüft und evt. kopiert werden, sondern es genügt oft schon, nur die jüngste Generation zu Bereinigen.

Darüberhinaus sind bei dieser Methode Allokationen auch sehr schnell. Da der komplette belegte Speicher immer einen zusammenhängenden Block bildet muss im Grunde nur der Zeiger auf das Ende des belegten Speicherbereiches inkrementiert werden.

Im Folgenden wird grob eine Mögliche Implementierung innerhalb der Unix Speicherverwaltung beschrieben.

Fazit: Bei Sprachen, die nur wenige Zuweisungen erzeugen (z.B. ML), ist diese Art von Garbage Collector sehr effizient. Die erhöhte Geschwindigkeit bei Allokationen kann den Zusatzaufwand der Speicherbereinigung fast komplett ausgleichen.

Review:

- Enthält eine gute, etwas detailliertere Beschreibung eines Generational Collectors.
- Der hier beschriebene Kollektor unterscheidet sich etwas vom moderneren in der .NET CLR, da er komplett auf dem Copying-Collector-Verfahren aufbaut.
- Die zweite Hälfte enthält weitestgehend Unix bezogene Implementierungsdetails und ist eher uninteressant.