IOOPM 2015

Comparing two methods of garbage collection

Karl Johansson

December 7, 2015

Contents

1 Abstract

Manual memory management is time consuming for programmers and a sourse for errors. There is a variety of different automatic memory management called garbage collectors that solves the issues one might have when handling the memory manually. But they come with different strengths and weaknessess. In this essay I have chosen to compare two different methods of garbage collection, the mark and sweep and the generational garbage collector. The mark and sweep algoritm, first marks all objects that are reacheable and still in use and then traverse the heap and collects all the objects that are not in use so the memory can be reused. The generational garbage collector is based on the weak generational hypothesis which states that most objects die young and old objects continue to live. The heap is devided into generations and the allocations of new objects are done on the first generation. The minor collections are done on the separate generations and if the objects is still in use it will get promoted to the next generation.

2 Introduction

Manual memory management means that the programmer is responsible for allocating the memory for the objects on the heap and when the object is no longer required to free the memory allocated on the heap so it can be recycled. This can be both time consuming and a source for errors. Errors like memory leaks caused by not freeing the memory allocated for the object when it will not be used any more and the error caused by prematurely freeing objects still in use[5]. Automatic memory management solves these issues as it handles the memory management and lets the programmer concentrate on the development of the programs. Automatic memory management is often referred to as garbage collection [4]. Presently there are many different garbage collection algorithms[4]. In this essay the mark and sweep algorithm and the generational garbage collection will be presented.

2.1 Mark-and-Sweep

The mark and sweep was the first garbage collection algorithm and it was part of the implementation of the programing language Lisp[6]. The goal of the algorithm is to mark all objects that are still in use as not garbage, so that the remaining objects which are not in use any more can be collected and the memory be reused.

To use the Mark-and-Sweep algorithm each object allocated on the heap is required to have an extra bit of memory to store the marker bit used by the algorithm. The marker bit is used to mark the object as not garbage[1].

The mark and sweep algorithm is divided into two phases. The first phase is the marking phase where all reachable objects are found and marked as not garbage. The reachable object are the objects that can be reached from the root set. The root set is the set of roots that are always accessible to the program. The algorithm follows all the pointers in the root set and marks the objects it finds as not garbage. The algorithm traverse each object and its pointers. If any marked object A has a pointer to another object B the algorithm follows that pointer to object B marks it as not garbage. The first phase is called the mark phase[6]. In the second phase the algorithm sweeps through the heap and frees all the objects that are not marked. This phase also unmarks all the objects that have been marked as not garbage in the first phase. This is done because the current garbage collection should not affect the next one. The second phase is called the sweep phase. The algorithm does not copy and compact the remaining objects on the heap. Which over time will lead to a fragmented memory. And it will make the allocation of a new object require more time to find space in the memory for it. In worst case it might not find the required space for the object[6].

2.2 Generational garbage collection

The Generational garbage collection makes use of the hypothesis called the weak generational hypothesis. Which states that most objects die young and old objects continue to live[9]. The goal of the generational is to separate the objects that will live longer from the short lived objects. It is unnecessary work to check the long lived objects during every minor collection.

The heap is divided into different parts called generations. In this example the heap will be divided into two generations. The first part is called the young generation and the other is called the old generation. New objects are allocated on the young generation. Each object

allocated on the heap is required to be able to store the amount of collections it has survived meaning each object will have extra bits of memory[8].

When the first generation reaches its threshold for garbage collection, a collection is performed only on the first generation and that collection is called a minor collection. As the collection is done on a smaller part of the heap it will be faster than say a Mark and sweep collection on the whole heap. If an object survives a collection the age of the object is increased. The age of the object is the number of collection it has survived[8].

Once an object has survived a predetermined amount of minor collections, the object is promoted and copied over to the old generation. This is done during a minor collection. The young generation will be smaller than the old generation because the minor collection should be fast. It needs to be fast as it will occur most frequently due to all new objects that are allocated on the young generation and it will also have the most garbage in form of the shortlived objects[8].

By promoting objects to the old generation the minor collection will not have to go traverse through each of those objects each time it does its collection. This means that the minor collection will avoid to repeatedly traverse long lived objects[3]. Once the old generation reaches its threshold for garbage collection a collection is done on the whole heap, that collection is called a major collection.

There are several different variations of the generational garbage collection. They vary in the amount of generations(more than in above example), the size of them and which method used for the collections.

3 Comparison

Runtime:

A garbage collector using mark and sweep will be activated less frequently than one using generational garbage collection because the objects are allocated one the whole heap and not on a smaller part as with the generational. But once the garbage collection is triggered it will take a longer time than that of the generational because it must search the whole heap. The generational will have more minor garbage collection but those will be a lot faster than those of the whole heap. And as the minor collections will collect most of the garbage it will take a longer time for the major collection to be triggered.

Fragmented memory:

The mark and sweep does not compact the memory, which will lead to fragmented memory[2]. Depending on the variation of the generational garbage collector, this might not be a issues as some use a mark and copy algorithm for the minor collection in the younger generation part of the heap. The younger generation is divided into three parts, the Eden where the new objects are allocated and two parts called survivor. Once the Eden is full, the minor collection goes through all the objects in the younger generation and those still reachable is copied to one of the survivor parts. Next time the Eden becomes full, the minor collection starts again and copies all reachable to the other survivor part. This means that every time the minor collection finds a reachable object it gets copied and the heap gets compacted in the survivor part. The Eden can be overwritten with new objects[7].

4 Summary

The development from manual memory management to automatic memory management feels natural, but to decide which is the best of the mark and sweep and the generational is not obvious. Which one is the best is depending on the application they will be used on. To use one does not exclude the usage of the other, like when the generational garbage collector does a major collection it might use the mark and sweep algorithm. With the Java HotSpot Virutal Machine you can use several different settings for the garbage collection, serial and parallel and copying and compacting[7].

After the research of both garbage collection methods, I get the impression that the generational garbage collectors are further development of the mark and sweep algorithm. A good solution for a garbage collector might be to use both the generational garbage collector with a copying and compacting algorithm on the minor collections and mark and sweep on the major collections.

5 References

References

[1] basen.oru.se. gc. http://basen.oru.se/kurser/koi/2008-2009-p1/texter/gc/index.html. visited on 2015-12-06.

[2] brpreiss. fragmentation problem. http://www.brpreiss.com/books/opus5/html/page425.html#SECTION00143100000000000000. visited on 2015-12-06.

[3] memorymanagement. generational-hypothesis. http://www.memorymanagement.org/glossary/g.html#term-generational-hypothesis. visited on 2015-12-06.

[4] memorymanagement. term-garbage-collection. http://www.memorymanagement.org/glossary/g.html#term-garbage-collection. visited on 2015-12-06.

[5] memorymanagement. term-manual-memory-management. http://www.memorymanagement.org/glossary/m.html#term-manual-memory-management. visited on 2015-12-06.

[6] memorymanagement. term-mark-sweep. http://www.memorymanagement.org/glossary/m.html#term-mark-sweep. visited on 2015-12-06.

[7] Oracle. fragmentation problem. http://www.oracle.com/technetwork/java/javase/memorymanagement-whitepaper-150215.pdf. visited on 2015-12-06.

[8] rose hulman. Generationalgc. http://www.rose-hulman.edu/Users/faculty/young/CS-Classes/csse490-dsr/200910/Slides/GenerationalGC.pdf. visited on 2015-12-06.

[9] David Ungar. Generation scavenging: A non-disruptive high performance storage reclamation algorithm. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.122.4295&rep=rep1&type=pdf. visited on 2015-12-06.