Garbage collection

Frank Wijmans

March 28, 2012

1 Memory management

Managing can be done manual or via garbage collection. The classical example is a free list. A free list is a data structure that allows for dynamic memory allocation, allowing (de)allocation very simple by taking free space from, and linking to a free space in the list. Reference counting allows you to count all references such that you can reallocate memory when it is not referenced. It is simple and lightweight, but also naïve and inefficient.

1.1 Mark-sweep

Mark-sweep algorithms first finds out what is free and what isn't to use this info later on. A variation on this algorithm is black-grey-white. Where black contains marked nodes with all pointers visited, grey nodes with some pointers visited and white nodes which are unvisited. The problem right now is running a program and garbage collection in parallel. It might cause for inconsistencies because pointers might change while some of those areas were already checked. It is solved by just taking all memory, and after that do garbage collection. The problem is that a program might stutter at such moment where it needs to clear up memory. In the next subsection even more strategies are stated, but c-like memory management becomes infeasible. It allows pointers to be modified like pointers.

1.2 Mark-compact

When using the mark-compact algorithm, you need moving memory, which results in no fragmentation nor administration. Moving memory is tricky and very expensive, while most problems are circumvented when using a language that doesn't contain expressions using pointers, moving memory all the time is to expensive. The copying garbage collector uses two halves of the memory. It uses one as a working part, and the second to de-fragment. It is fast and particularly good for short-lived objects. Problems with cycles are solved by replacing the old address with a link to the new address. A solution to get it working with long lived objects, is to save those in a third part of the memory. The problem is that you don't yet know when objects are long lived. When observing the lifespan of an object you can generate a garbage collector.

1.3 Generational garbage collector

Each object gets an age, and gets put in the right area of the memory. The bigger or long lived the objects are, the smaller the chance that it gets collected as garbage. The smaller or short lived, it gets collected earlier. Not good for real time systems, but good for interactive systems. This does pose problems like: finalization, stable and weak pointers.