Automatisk skräphantering

Ulf Sigvardsson

November 3, 2017

Contents

1 Inledning

Inom datavetenskapen beskriver termen skräphanterare, eller *garbage collector*, processen att (i ett programmeringsspråk) spåra antalet referenser till ett visst objekt i minnet. Syftet är att korrekt kunna bedöma när ett objekt kan anses onåbart och dess plats i minnet är ledig. Automatiseringen av minnefrigöring är en vital del för korrekta program och effektiv programmering. Det existerar en rad olika implementationer, alla med sina fördelar respektive nackdelar, och här beskrivs och jämförs ett urval av dessa tekniker

2 Olika typer av skräphanterare

I denna sektion beskrivs implementationerna av tracing garbage collectors samt reference counting garbage collectors.

2.1 Tracing Garbage Collectors

Med samlingsnamnet *tracing garbage collectors* avses skräpsamlare som på ett eller annat sätt avgör ett objekts status som nåbar genom en referenskedja från s.k. rotvariabler. Dessa variabler är vanligtvis stackvariabler, globala variabler och funktionsargument.

2.1.1 Mark-sweep GC

Metoden *mark-sweep* ålägger skräpsamlaren att, när programmet får ont om ledigt minne på heapen, frigöra objekt som inte längre är nåbara för programmet. Sådana objekt kan röra sig om omdirigerade pekare eller stackvariabler från ett tidigare funktionsanrop vars scope programmet nu har lämnat. Således sker ingen kontinuerlig skräphantering precis när ett objekt blir onåbart, utan körs vid behov.

Varje objekt håller en bitflagga för att indikera om det refereras av något annat objekt eller inte. Om denna flagga är satt finns det ett objekt (som är kan nås från en rot) som refererar till det och per defenition är det nåbart av programmet. Om inte är objektet kvalificerad för borttagning.

När skräpsamlaren körs inleder den med den s.k. *markeringsfasen*. Under denna fas itererar skräpsamlaren över mängden av rötter. För var och en av dessa rötter markeras alla objekt som roten direkt referar till som nåbara, och detsamma för alla objekt som dessa objekt refererar till, etc. När markeringsfasen är över har således varje objekt som kan nås från någon rot markerats som nåbar [3].

Därefter följer *sweepfasen*, där hela minnet söks igenom och objekt som inte markerats som använda frigörs. Objekten, vars bitflagga initierades under markeringsfasen, blir nu nollställda. Detta möjliggör en ny utvärdering vid

kommande körning[3].

Nackdelen med mark-sweep implementationen är att programmet måste avbrytas. Om programmet inte avbryts finns risken att objekt allokeras, förlorar referenser eller på andra sätt muteras under tiden skräpsamlaren arbetar. Eftersom den här sortens skräpsamlare körs då det finns ont om ledigt minne sker detta på ett oförutsägbart manér och passar sig särskilt dåligt i interaktiva program eller där tidsprecision är av vikt i största allmänhet.

2.1.2 Tri-color marking

Förutom att kontrollera samtliga stackvariabler och dess inre referenser, måste, i en mark-sweep-implementation, också hela arbetsminnet itereras över för att frigöra minne. Eftersom fler än en rot kan referera till samma objekt finns också risk för dubbelt arbete.

Tri-color-marking undviker sådant extraarbete genom att dela in alla heap-objekt i tre delmängder.

- Den *grå mängden* representerar objekt som kan frigöras
- Den *vita mängden* innehåller objekt som är nåbara från en rot och inte refererar till några objekt i den vita mängden
- Den *svarta mängden* består av objekt som är nåbara från en rot med inte ännu har undersökts för referenser till vita objekt

SKräpsamlaren fastställer vilka objekt som ska frigöras genom att låta den svarta mängden vara inledningsvis tom och den grå mängden innehålla alla objekt som refereras av någon rot. Den vita mängden utgör alla andra objekt.

Därefter väljs ett grått objekt som sätts till svart, och alla *vita* objekt som refereras av det flyttas till den grå mängden. Algoritmen itererar över dessa två steg tills dess att inga grå objekt finns[1].

En effekt av algoritmen är att inget svart objekt kan någonsin referera till ett vitt objekt, således kan alla vita objekt frigöras så snart den gråa mängden är tom. Tri-color marking behöver inte suspendera programmets körning utan kan användas kontinuerligt genom att markera objekt då de allokeras eller redigeras, närhelst den vita mängden är tom kan minne således frigöras.

2.2 Reference Counting Garbage Collectors

En skräpsamlare som använder sig utav *reference counting* håller information om hur många referenser till ett visst objekt som finns i ett program. Varje gång en referens till ett objekt eller minneblock skapas inkrementeras antalet referenser till detta objekt. Analogt dekrementeras antalet referenser varje gång en referens pekas om eller faller utanför scope, t.e.x. efter att en funktion har slutfört sitt arbete.

En fördel med denna teknik är att ett objekt kan frigöras så snart inga referenser till det längre finns i programmet eftersom det då per defenition är onåbart. När ett objekt frigörs minskas även antalet referenser till alla objekt som refereras från det, därmed kan stora mängder utrymme frigöras vid ett och samma tillfälle.

Denna teknik har även fördelen av att inte behöva avbryta programmets körning för att eventuellt frigöra minne. Detta eftersom enskilda objekt kan frigöras när deras referensräknare når noll.

En av nackdelarna med en sådan implementation jämfört med s.k. *tracing garbage collectors* är att den inte täcker fallet med cirkelreferenser. Ett objekt i en referenskedja, exempelvis ett objekt som refererar till sig självt, kommer således alltid att ha minst ett objekt som refererar till sig och blir aldrig en kandidat för borttagning. På grund av detta tillkortakommande används inte referensräknande skräpsamlare fristående utan i kombination med någon annan typ av skräpsamlare, exempelvis mark-sweep, som med jämna mellanrum anropas för att hantera cirkelreferenser[2].

Eftersom referensräknande skräpsamlare körs i mindre inkrement anropas de oftare än tracing GC:s vilket ger upphov till viss ineffektivitet. Dessutom krävs att varje objekt reserverar plats för ytterligare metadata i form av antal referenser till objektet vilket leder till ökad minnesanvändning.

3 Slutsats

Tracing-, och reference counting-skräpsamlare innefattar endast en liten del av de tillgängliga varianterna. Klart står dock att de båda har tillkortakommanden och fördelar gentemot varandra.

Alla program som kräver hög tidsprecision omöjliggör i stort sett användning av en renodlad mark-sweep-skräpsamlare då avbrotten i programmets körning blir för kostsamma. Detta problem finns inte hos tri-color-marking-skräpsamlare som kan köras periodvis och inte vid behov.

Referensräknande skräpsamlare är mer flexibla än de tidigare nämnda efter-

som enskilda objekt granskas och frigörs kontinuerligt, dock krävs att någon typ av tracing-skräpsamlare används för att ta hand om cirkelreferenser så i metoden kan inte stå på egna ben. Dessutom finns möjligheten att ytteligare objekt rekursivt frigörs när ett objekts referensräknare slår noll, så realtids-frigörningen sker inte alltid som väntat.

Det är denna författares uppfattning att av de olika implementationerna i denna text är tri-color-marking den mest förutsägbara gällande beteende och prestanda och är den mest kompletta i en självständig implementation.

References

[1] Mark S. Johnstone, Paul R. Wilson, *The Memory Fragmentation Problem: Solved?*, 1998.

[2] Abhinaba Banu, Back To Basics: Reference Counting Garbage Collection *https://blogs.msdn.microsoft.com/abhinaba/2009/01/27/back-to-basics-reference-counting-garbage-collection/*

[3] Chirag Agarwal, Mark-and-Sweep garbage Collection Algorithm *http://www.geeksforgeeks.org/mark-and-sweep-garbage-collection-algorithm/*