TÖL304G Forritunarmál
Um ruslasöfnun

Snorri Agnarsson

20. október 2024

Efnisyfirlit

1 Ruslasöfnun — Garbage Collection

Ætlast verður til að þið kunnið skil á eftirfarandi ruslasöfnunaraðferðum, að undanskilinni þeirri síðustu, þ.e. *generation scavenging*.

You will be expected to know the following garbage collection methods, except the last one, i.e. *generation scavenging*.

1.1 Reference Count

Þessi aðferð, sem kalla má tilvísunartalningu á íslensku felst í því að í hverju minnissvæði í kös er teljari, sem ávallt inniheldur fjölda beinna tilvísana á þann hlut (það minnissvæði) úr breytum í forritinu eða öðrum hlutum[1]. Þegar þessi teljari verður núll má skila minnissvæðinu.

With this method, called *reference count*, each memory area contains a counter, which always contains the number of direct references to to that object (i.e. that memory area) from variables in the program or from other objects[2].

[1] Athugið að hér er um *fastayrðingu gagna* að ræða.

[2] Note that this is a *data invariant*.

1.2 Mark and Sweep

Þessi aðferð vinnur þannig að ónotuð minnissvæði eru geymd í lista, sem kallaður er *frjálsi listinn*. Þegar reynt er að úthluta minni er athugað hvort eitthvað er tiltækt á frjálsa listanum. Ef svo er ekki þarf að ruslasafna. Sérhvert minnissvæði s þarf að hafa einn bita, s.merki, sem er notaður til merkingar á meðan á ruslasöfnun stendur. Reiknað er með því að sérhvert minnissvæði sé ómerkt þegar ruslasöfnun hefst.

Þegar þarf að ruslasafna er eftirfarandi `safna` stef framkvæmt:

```
stef safna()
{
  ;;; merkja (mark):
  fyrir sérhverja breytu b í forritinu
    merkja(b)
  ;;; sópa (sweep):
  fyrir sérhvert minnisvæði s í kösinni
    ef s.merki er ekki satt
      bæta s á frjálsa listann
    annars
      s.merki = ósatt
}

stef merkja(b)
{
  ef b.merki er ekki satt (ekki sett)
  {
    b.merki = satt
    ef b vísar á hlut sem hefur tilvísanir út
      fyrir sérhvert sæti (eða tilviksbreytu) i í b
        merkja(b[i])   ;;; eða: merkja(b.i)
  }
}
```

With this method, unused memory areas are stored in a list called the *free list*. When an attempt is made to allocate memory the free list is checked to see whether a memory area is available there. If so, there will be no need yet for garbage collection. Each memory area s needs to have one bit s.mark, that is used to mark the area while garbage collecting. It is assumed that each memory area is unmarked when garbage collection is initiated.

When garbage collection is needed, the following `collect` function is executed:

```
function collect()
{
```

```
  ;;; Mark:
  for each variable b in the program
    mark(b)
  ;;; Sweep:
  for each memory area in the heap
    if s.mark is not true
      add s to the free list
    else
      s.mark = false
}

function mark(b)
{
  if b.mark is not true (not set)
  {
    b.mark = true
    if b refers to an object that refers to other memory areas
      for each position (or instance variable) i in b
        mark(b[i])   ;;; or: mark(b.i)
  }
}
```

1.3 Stop and Copy

Þessa aðferð má kalla afritunaraðferðina á íslensku. Hún felst í því að nota tvö jafnstór minnissvæði fyrir kös, og er aðeins annað svæðið í notkun hverju sinni (nema meðan á ruslasöfnun stendur). Minni er ávallt úthlutað aftast í því svæði sem er í notkun.

Í byrjun ruslasöfnunar er víxlað á svæðum og öll svæði í gamla svæðinu sem eru í notkun eru afrituð í nýja svæðið. Reiknað er með því að sérhvert minnissvæði s hafi einn bita s.merki fyrir merki og sá biti er ekki notaður í neitt annað. Einnig er reiknað með því að sérhvert minnissvæði hafi pláss fyrir framvísunarbendi, s.framvísun, en það pláss má nota í annað þegar ruslasöfnun er ekki í gangi. Minnissvæði sem lifa af ruslasöfnun eru afrituð yfir í nýju kösina og eru þá um leið merkt og fá framvísunarbendi sem bendir á samsvarandi minnissvæði í nýju kösinni. Eins og í *merkja og sópa* aðferðinni er reiknað er með því að öll minnissvæði séu ómerkt þegar ruslasöfnun hefst.

```
stef safna()
{
  víxla kösum
  fyrir sérhverja breytu b í forritinu
    b = afrita(b)
```

```
}

stef afrita(b)
{
  ef b vísar í nýju kösina eða er frumstætt gildi
    skila b
  ef b.merki er satt (b hefur þegar verið afritað)
    skila b.framvísun
  notum staðværar breytur nýttb og f
  nýttb = nýtt svæði í nýju kösinni, jafnstórt b, eins og b
  f = b.framvísun
  b.framvísun = nýttb
  b.merki = satt
  nýttb.framvísun = afrita(f)
  fyrir sérhvert annað sæti k í b (annað en b.framvísun)
    nýttb.k = afrita(b.k)
  skila nýttb
}
```

With the stop and copy method we have two equally large heaps, only one of which is used at any time, except while garbage collecting. The heap in use is split into two contiguous areas, one of which contains memory in use, and the other one with currently unused memory. When memory is allocated, the unused area shrinks and the used area increases.

When allocation is attempted and it turns out that the unused area of the current heap is not large enough for the allocation, we need to garbage collect. When garbage collection is started, we switch the two heaps and subsequent allocation will be from the new heap, which is initially empty. Garbage collection then consists of copying all the memory areas that are reachable in the old heap into the new heap. It is assumed that each memory area s has one bit, s.mark for marks and that bit is not used for anything else. Also, it is assumed that each memory area is large enough to contain one reference (pointer), s.forward, to another memory area. The memory area s.forward may be used for other information when not garbage collecting. Objects survive garbage collection, if they are reachable and therefore get copied. When the memory area is copied, it is marked and a forwarding address is written into them that refers to the new memory for the object in the other heap. As in *mark and sweep* we assume all memory areas are unmarked when garbage collection starts.

```
function collect()
{
  swap heaps
  for each variable b in the program
```

```
    b = copy(b)
}

function copy(b)
{
  if b refers to the new heap or is a primitive value
    return b
  if b.mark is true (b has already been copied)
    return b.forward
  use local variables newb and f
  newb = new area in the new heap, as large as b, like b
  f = b.forward
  b.forward = newb
  b.mark = true
  newb.forward = copy(f)
  for each other position k in b (other than b.forward)
    newb.k = copy(b.k)
  return newb
}
```

1.4 Generation Scavenging

Ofangreindar þrjár aðferðir eru sígildar ruslasöfnunaraðferðir, sem þið verðið að kunna skil á. Nýjar og miklu betri aðferðir hafa verið að ryðja sér til rúms, m.a. kynslóðasöfnunaraðferðir svo sem *generation scavenging*.

Hún virkar þannig að notað er afritunaraðferð að hluta til, en minnissvæði, sem lifa af margar kynslóðir (ruslasafnanir) fá fastráðningu (*tenure*), og eru flutt í sérstaka kös fyrir fastráðin svæði. Fastráðnum minnissvæðum er aldrei skilað, sama hve gagnslaus þau eru orðin (þ.e. þótt engin tilvísun sé lengur til í viðkomandi svæði).

Reyndar er yfirleitt hægt að gera sérstaka ruslasöfnun á fastráðin svæði, en þá er kerfið oft ónothæft á meðan á því stendur, enda venjulegar fastayrðingar brotnar.

Fyrir þá sem áhuga hafa á að lesa meira um ruslasöfnun eru heilmiklar upplýsingar til á vefnum, til dæmis á vefsíðu Richard Jones[3]. Einnig er yfirlitsgrein eftir Paul R. Wilson á vefnum[4].

The above three methods are classic garbage collection methods, which you must be familiar with. Newer and much better methods have come along, for example generational methods such as *generation scavenging*.

That method partly uses stop and copy, but with the twist that objects that survive many collections get tenure and are moved into a separate heap for tenured objects

[3] http://www.cs.ukc.ac.uk/people/staff/rej/gc.html

[4] https://notendur.hi.is/snorri/downloads/bigsurv.pdf

and will never be considered for deletion after that, however useless they may become, even if there are no longer any references to the object.

Actually it is usually possible to do a special garbage collection of the tenured heap, but during such a garbage collection the system is unusable as the usual data invariants are broken.

For those who may be interested in reading more about garbage collection there is a great deal of information on the web, for example on Richard Jones' web page[5]. Also there is a survey article by Paul R. Wilson on the web[6].

[5] http://www.cs.ukc.ac.uk/people/staff/rej/gc.html

[6] https://notendur.hi.is/snorri/downloads/bigsurv.pdf