Факултет Техничких Наука
Универзитет у Новом Саду

Виртуелизација процеса

Garbage Collector (GC)

Large Object Heap (LOH) and GC types

Нови Сад, 2023.

Internal

Large Object Heap (LOH)

- Објекти живе у меморијским сегментима *heap*-а

- Када се *CLR* учита, *GC* затражи од оперативног систма два почетна сегмента:
  - *Small Object Heap - SOH*
  - *Large Object Heap – LOH*

- Меморијски сегмент је дио меморије који *GC* резервише од оперативног система позивањем функције *VirtualAlloc*

- *LOH* се често назива “генерација 3”

Internal

Large Object Heap (LOH)

- Алокација и аутоматско сакупљање смећа на *LOH*-у је прилично сложен процес
- Сви објекти већи или једнаки 85KB се смештају на *LOH*
- Тешко је имати објекат ове величине, углавном су то низови и колекције
- Сабијање ових објеката, тј. њихово копирање на неку другу слободну локацију може бити веома “скупо”
- Главни разлог због чега имамо *LOH* је да би над свим великим објектима применили исти алгоритам аутоматског управљања меморијом у циљу смањења утицаја на перформансе

Internal

Large Object Heap (LOH)

- Инжењер пишући код може да "додаје" елементе у генерацију 0 и на *LOH*
- Само *GC* може да "додаје" елементе у:
  - Генерацију 1 – преживели из гнерације 0
  - Генерацију 2 – преживели из генерације 1
- Када се покрене *GC*, он брише објекте који се не користе са *LOH*-а, остављајући за собом фрагментован сегмент
- Дефрагментација *LOH*-а је скупа!
- Шта је алтернатива?

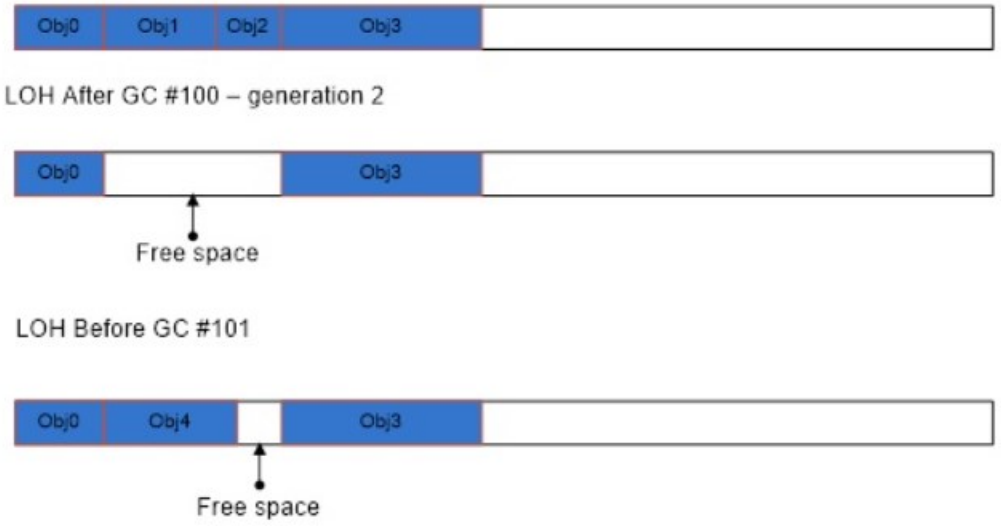
Internal

Покретање *GC*-а на *LOH*-у

- Мала је вероватноћа да ће објекти које треба да додамо бити исте величине као што је слободан простор настао након фрагментације
- Као резултат тога, мали делови меморије ће увек остати између објеката
- Ово доводи до фрагментације
- Ако су делови слободног простора мањи од *85 KB* никада се неће попунити, јер такви објекти не стижу на *LOH*

Internal

Пример *GC*-а на *LOH*-у



Покретање *GC*-а на *LOH*-у

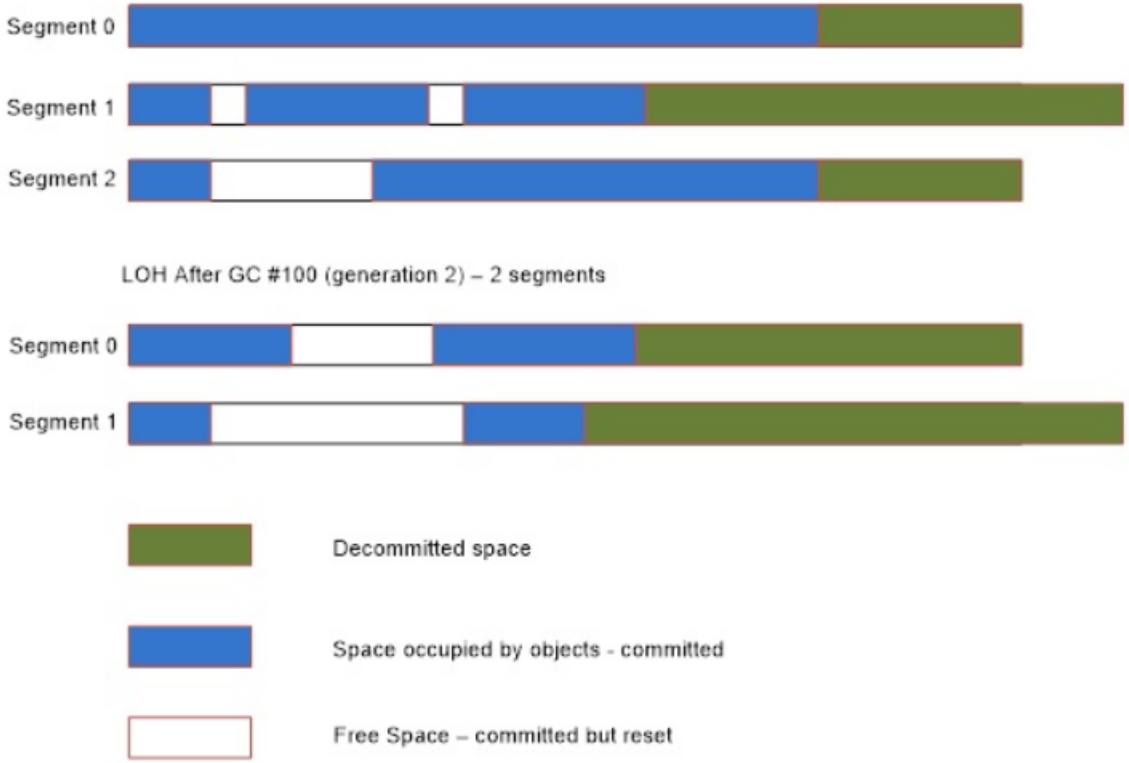
- Ако нема довољно простора да се смести објекат на *LOH, GC* може да уради следеће:
  - Потражи адекватан слободан простор у фрагментисаним деловима
  - Тражи додатну меморију од оперативног система
  - Покрене анализу генерације 2 у нади да ће се нешто ослободити

- Боље је ићи на другу опцију ако прва не прође, јер је покретање потпуног *GC* скупо

- Након што се временом ослободи простор, *GC* може позивом функције *VirtualFree* да ослободи сегменте назад у оперативни систем које је раније од њега тражио

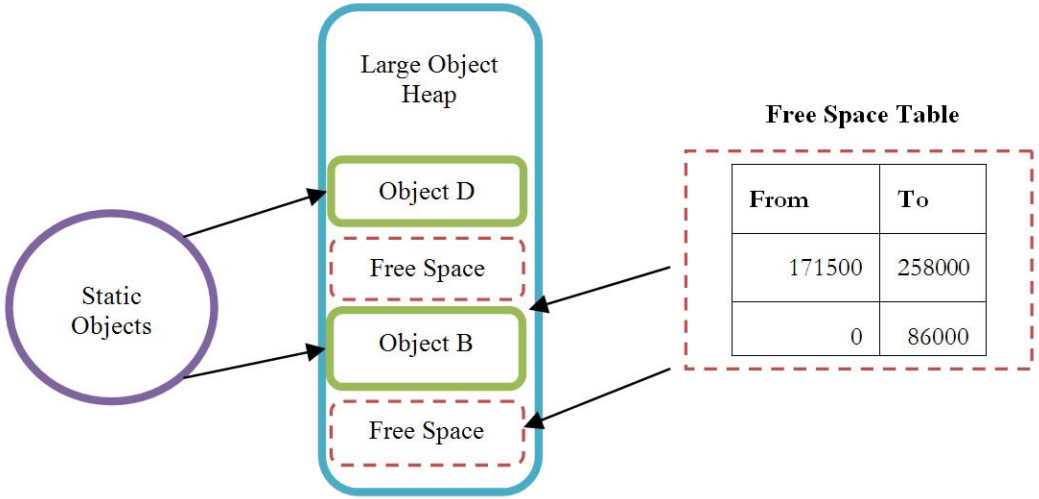
Internal

Ослобађање сегмената назад у ОС



Табела слободних места

- Инфорамација о свим слободним деловима меморије на једном месту

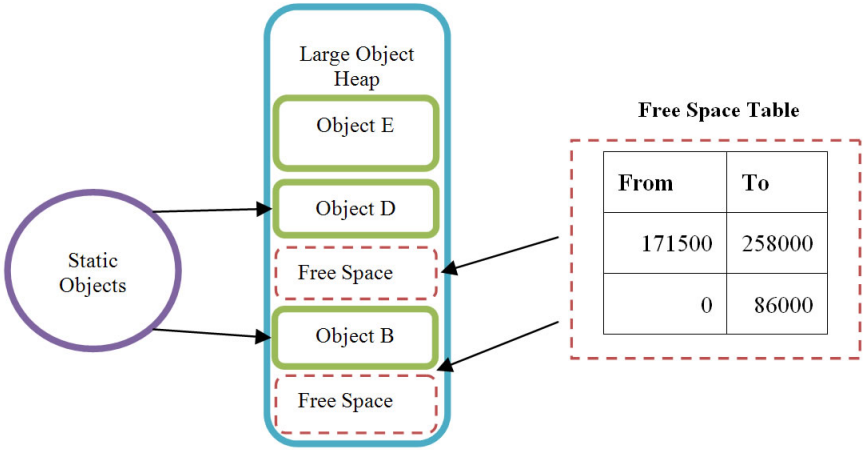


Internal

Табела слободних места

- Приликом доделе новог објекта на *LOH*, проверава се у табели слободног простора да ли има адекватно место

- Ако постоји, објекат се додаје на почетак адресног простора и смањује се фрагментација, а ако нема додаје се на следећи слободан простор као на слици



Internal

Утицај на перформансе

- Алокација на *LOH*-у:
  - *CLR* мора да гарантује да ће бити меморије за сваки нови објекат, тако што мора да ослободи простор ако га нема
  - Брисање једног бајта траје 2 циклуса, што значи да за један велики објекат треба 170к циклуса
  - Брисање меморије од 16 *MB* на 2-*GHz* машини траје 16 *ms*

- Деалокација на *LOH*-у:
  - *LOH* и генерација 2 се прикупљају заједно, и ако се прекорачи било који од два прага, креће прикупљање
  - Ако и генерација 2 и *LOH* имају доста података, утицај на перформансе може бити значајно велики

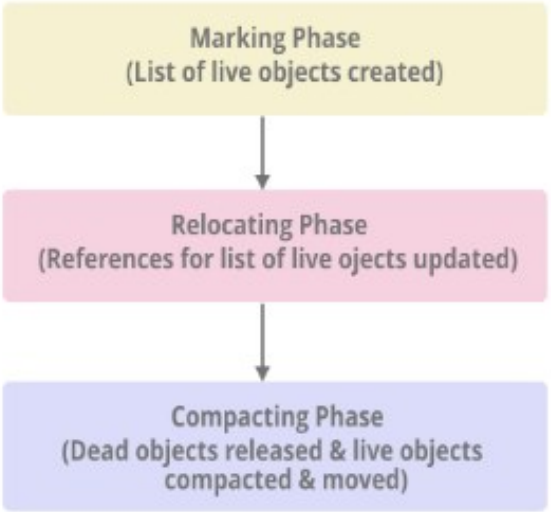
Internal

Важне чињенице

- Битно је да схватите важност животног века објеката
- Где и када се креира објекат је кључно за вашу апликацију
- Додавање референце са једног објекта на други, било да се ради о колекцијама, догађајима, делегатима, вероватно ће одржаати објекат живим дуже него што мислите
- Те референце ће их убацити у генерацију 2 веома брзо
- Видели сте какве изазове носи са собом генерација 2
- На крају апликација заузима више него што је величина свих објеката

Internal

Фазе и типови *GC*-а



- Постоје два основна типа рада *GC*-а:
  - *Workstation garbage collection* – дизајниран за клијентске апликације
  - *Server garbage collection* – дизајниран за серверске апликације

Internal

Workstation garbage collection

- *GC* у овом режиму рада пружа максимални одзив корисничком интерфејсу
- Овај тип GC-а се користи да се обезбеди *throughput*
- Основна идеја је да се предност да извршавању у апликацији него руковању меморијом
- Сакупљање се извршава на једној нити
- Сакупљање може бити конкурентно са извршним нитима

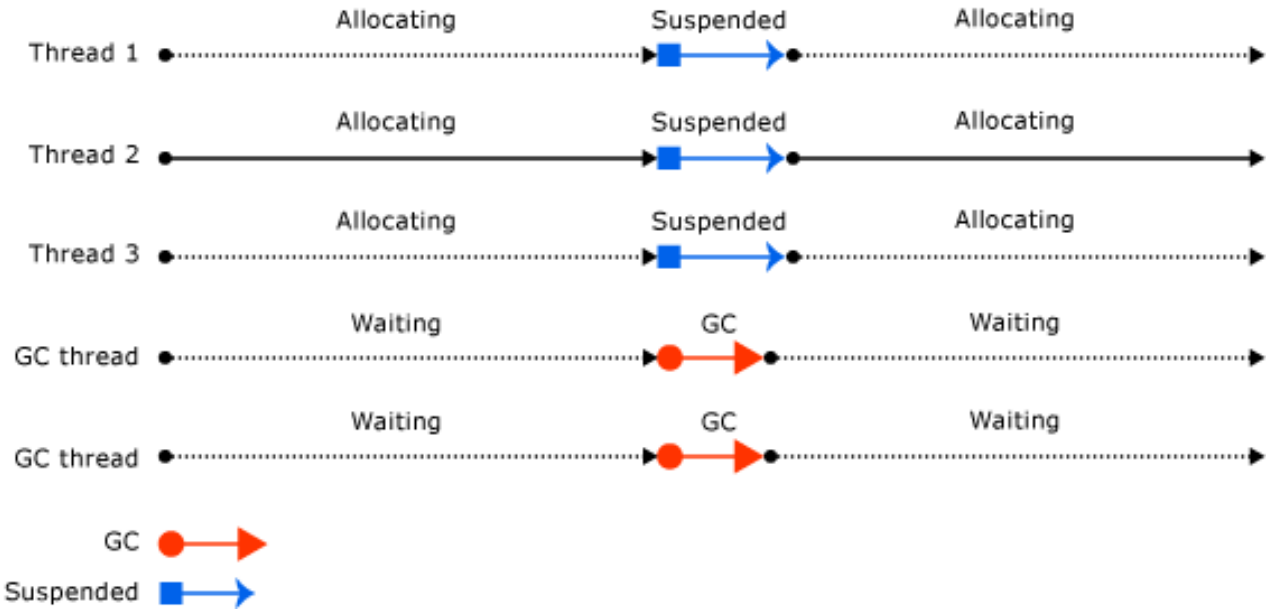
Internal

Server garbage collection

- Начин рада *GC*-а је прилагођен у циљу давања максималне пропусности и скалабилности ваше апликације
- Овај тип *GC*-а треба да пружи континуирано одржавање и управљање меморијом без ометања перформанси апликације
- Сакупљање смећа се дешава на више наменских нити које су покренуте на *THREAD_PRIORITY_HIGHEST* нивоу приоритета
- Са обзиром да овде имамо више нити које раде у паралели, прикупљање смећа је брже од прикупљања смећа на радној станици

Internal

Server garbage collection



Конфигурисање *GC-a*

- Да би омогућио *Server GC* потребно је да урадите следеће *gcServer="true"*

```
<configuration>
   <runtime>
      <gcServer enabled="true | false"/>
   </runtime>
</configuration>
```

- Ако поставимо на *false* тај начин ћемо омогућити *Workstation GC*

Internal

Конфигурисање *GC-a*

- Ако је подешен *GC* на Workstation онда се може укључити конкуренти мод на следећи начин:

```
<configuration>
   <runtime>
      <gcConcurrent enabled="true | false"/>
   </runtime>
</configuration>
```

Internal

Runtime GC Latency Control

- *GCLatencyMode.Batch* – дизајниран да пружи максималну пропусност и перформансе за **делове** апликације где одзив корисничког интерфејса није важан

- *GCLatencyMode.LowLatency* – смањује утицај *GC*-а на минимум што је добро у случајевима када је важан одзив корисничког интерфејса

- *GCLatencyMode.Interactive* – *Workstation GC* са укљученим конкурентним модом, дајући баланс између *GC* ефикасности и одзива апликације

Internal

Runtime GC Latency Control

- Очигледна употреба *LatencyMode*-а је да се промени на кратак период током извршавања критичног кода којем је потребан максимални кориснички интерфејс или перформансе групне обраде и онда се поново врати

```
using System.Runtime;
…
// Store current latency mode
GCLatencyMode mode = GCSettings.LatencyMode;
// Set low latency mode
GCSettings.LatencyMode = GCLatencyMode.LowLatency;
try
{
   // Do some critical animation work
}
finally
{
   // Restore latency mode
   GCSettings.LatencyMode = mode;
}
```

Internal

GC Notification

- *.NET Framework* вам дозвољава да знате када ће се извршити комплетан *GC* како би се предузели неопходни кораци

- *.NET Framework* вам у суштини омогућава следеће:
  - Региструјете се да примате обавештења о комплетном *GC*-у
  - Одредите када се ради комплетан *GC*
  - Када се завршио
  - ...

```
GC.RegisterForFullGCNotification(10, 10);
```

Internal

GC Notification

```
System.Collections.ArrayList data = new ArrayList();
bool carryOn = true;
private void b1_Click(object sender, EventArgs e)
{
   GC.RegisterForFullGCNotification(10, 10);
   Thread t = new Thread(new ThreadStart(ChecktheGC));
   t.Start();
   while (carryOn)
   {
      data.Add(new byte[1000]);
   }
   GC.CancelFullGCNotification();
}
```

```
private void ChecktheGC()
{
 while (true) // Wait for an Approaching Full GC
 {
    GCNotificationStatus s = GC.WaitForFullGCApproach();
    if (s == GCNotificationStatus.Succeeded)
    {
       Console.WriteLine("Full GC Nears");
       break;
    }
 }
 while (true) // Wait until the Full GC has finished
 {
   GCNotificationStatus s = GC.WaitForFullGCComplete();
   if (s == GCNotificationStatus.Succeeded)
   {
     Console.WriteLine("Full GC Complete");
     break;
   }
 }
 carryOn = false;
}
```

Object Pinning and GC handles

- До сада смо тврдили да је управљање меморијом боље на *SOH*-у него на *LOH*-у. Због чега?
- Због проблема са фрагментацијом код *LOH*-а!
- Међутим, ако желимо да упућујемо позиве другим *API*-ма *unamanged* апликација, вероватно ћемо им прослеђивати податке
- Ако су подаци на *LOH*-у, могуће да ће се преместити некада током његовог сабијања
- Ово није проблем за *.Net* апликације, али јесте за за *unamanged* које се ослаљају на фиксне локације за објекте

Internal

GC handles

- Потребан нам је начин за боље руковање објектима које размењујемо између *manged* и *unamanaged* домена

- *.Net* користи структуру *GCHandle* за праћење објеката на *heap*-u

- *.Net* одржава табелу *GCHandles* да би то постигао
  - *Normal* – прати стандардне *heap* објекте
  - *Pinned* – фиксира објекте на одређеној адреси у меморији

Internal

Тема за наредно предавање

Delegate-и и *Event*-и

Видимо се и хвала!

Internal