

# Re: How GC Works

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.clr/2005-04/msg00068.html>

---

- *From:* [saurabhgarg2k@xxxxxxxxxx](mailto:saurabhgarg2k@xxxxxxxxxx)
  - *Date:* 8 Apr 2005 07:09:03 -0700
- 

Hi Lyon,

Thanks for the Reply, but actually there is no problem with the Code, actually its a sample code, what i was trying to understand that the author of the code is saying that First For Loop works slower than second and it does, i was just trying to understand how. His Explanation is:

In the First For loop, there is only one class which implements Finalize method and since the Class With finalize method requires at least two GC Cycles to clean up the memory, so result is that the huge amount of memory it allocates will not get cleared up in one GC Cycle and so managed heap will be full again soon which makes GC runs too frequently and makes application slower.

But in the second for loop he separates the Class in two Classes, one with Finalize and the other with huge memory, so his point is that now the Class which allocates huge memory does not implement Finalize method and will get cleared up in one GC cycle and deallocates the huge memory which in turn results in less GC runs and application runs faster.

Now what my doubt is:

As we know that GC will happen after managed heap is full (but as u said it might not be the case), but let's say it does, so in the first case no objects will be deallocated because all objects in the heap are of the type which implements Finalize method, and they need at least two GC cycles, so if no memory is freed, where from the CLR gets the space to give it to the new object. Will GC run again and again unless the memory gets sufficiently cleared for the new object.

Looking forward for a reply

Thanks and Regards  
Saurabh Garg

"Chris Lyon [MSFT]" wrote:

Re: How GC Works

> Hi Saurabh  
>  
> A couple of points:  
>  
> -There are several situations that will trigger the GC to perform a  
> collection, most of which occur before the entire heap is full.  
Maoni has  
> some great explanations at <http://blogs.msdn.com/maoni/>  
>  
> -If your class uses unmanaged resources, it should use the Dispose  
Pattern,  
> and not rely on finalization. This will solve the problem of your  
object  
> being promoted a generation due to being put on the finalization  
queue.  
>  
> ([http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/h  
tml/cpconfinalizedispose.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconfinalizedispose.asp))  
>  
> More information about finalization, GC and Performance:  
> <http://blogs.msdn.com/ricom/>  
> <http://blogs.msdn.com/cbrumme>  
> <http://blogs.msdn.com/clyon>  
>  
> Hope that helps  
> -Chris  
>  
> -----  
>  
> | Hi there,  
> |  
> | I am attaching a snippet of code here which tries to explain that  
if u  
> | have a class which contains both Unmanaged resources and allocates  
a  
> | lot of memory and have a finalize method, then u should divide it  
> | into two classes, one contains the memory part and other manages  
the  
> | unmanaged resources and implement finalize in the second class so  
that  
> | the memory will be released quick enough for first Class which  
> | allocates large memory as classes implementing finalize method  
requires  
> | at least 2 GC cycles to release the objects.  
> |  
> |  
> | Here is the first class which contains both large memory and  
Unmanaged  
> | resources in one class  
> |  
> | Class SimpleObject

## Re: How GC Works

```
> | Private Declare Function CreateSemaphore _
> | Lib "kernel32" Alias "CreateSemaphoreA" _
> | (ByVal lpSemaphoreAttributes As Integer, _
> | ByVal lInitialCount As Integer, _
> | ByVal lMaximumCount As Integer, _
> | ByVal lpName As String) As Integer
> | Private Declare Function ReleaseSemaphore _
> | Lib "kernel32" Alias "ReleaseSemaphore" _
> | (ByVal hSemaphore As Integer, _
> | ByVal lReleaseCount As Integer, ByVal _
> | lpPreviousCount As Integer) As Integer
> |
> | Dim arr() As Integer
> | Dim handle As Integer
> |
> | Sub New()
> | ReDim arr(10000) ' a lot of memory
> | ' directly allocate an unmanaged resource
> | handle = CreateSemaphore(0, 1, 1000, _
> | "testsemaphore")
> | End Sub
> |
> | Protected Overrides Sub Finalize()
> | ' release the unmanaged resource
> | If handle <> 0 Then _
> | ReleaseSemaphore(handle, 1, 0)
> | End Sub
> | End Class
> |
> |
> | Here is the Next Class which divides the functionality which is
> | embedded in the outer class.
> |
> | Class CompoundObject
> | Dim arr() As Integer
> | Dim wrapper As SemaphoreWrapper
> |
> | Sub New()
> | ReDim arr(10000) ' a lot of memory
> | ' allocate another (managed) resource
> | ' (no need to release it in Finalize)
> | wrapper = New SemaphoreWrapper
> | End Sub
> |
> | ' a class that wraps the unmanaged resource
> | Private Class SemaphoreWrapper
> |
> | Private Declare Function CreateSemaphore _
> | Lib "kernel32" Alias "CreateSemaphoreA" _
> | (ByVal lpSemaphoreAttributes As Integer, _
> | ByVal lInitialCount As Integer, _
```

## Re: How GC Works

```
> | ByVal IMaximumCount As Integer, _
> | ByVal lpName As String) As Integer
> |
> | Private Declare Function ReleaseSemaphore _
> | Lib "kernel32" Alias "ReleaseSemaphore" _
> | (ByVal hSemaphore As Integer, _
> | ByVal lReleaseCount As Integer, _
> | ByVal lpPreviousCount As Integer) As Integer
> |
> | Public Handle As Integer
> | Sub New()
> | ' allocate the unmanaged resource
> | Handle = CreateSemaphore(0, 1, 1000, _
> | "testsemaphore")
> | End Sub
> |
> | Protected Overrides Sub Finalize()
> | ' release the unmanaged resource
> | If Handle <> 0 Then _
> | ReleaseSemaphore(Handle, 1, 0)
> | End Sub
> |
> |
> | End Class
> | End Class
> |
> | Here is the Client Code
> |
> | Module Module1
> | Sub Main()
> | Dim i As Integer
> | Const TIMES As Integer = 10000
> |
> |
> | Dim t As Date = Now
> | For i = 1 To TIMES
> | Dim so As New SimpleObject
> | Next
> | Console.WriteLine("Simple object: {0}", _
> | Now.Subtract(t))
> |
> |
> | ' force a garbage collection to clean up
> | GC.Collect()
> | GC.WaitForPendingFinalizers()
> |
> |
> | t = Now
> | For i = 1 To TIMES
> | Dim co As New CompoundObject
> | Next
```

## Re: How GC Works

```
> | Console.WriteLine("Compound object: {0}", _
> | Now.Subtract(t))
> | Console.ReadLine
> | End Sub
> | End Module
> |
> |
> | Now the author claims that the first for loop will take more time
as it
> | requires more GC cycles to deallocate the large chunk of memory it
> | allocates and the second will release it quickly as the class
> | allocating the Large memory does not implement finalize method so
it
> | will be deallocated in one GC cycle unlike the class with unmanaged
> | resource.
> |
> |
> | My question:
> | As GC will only fire when managed heap runs out of space, so in
first
> | for loop when after allocating a few objects, heap is full and the
GC
> | fires but no objects will be allocated as the class implements the
> | Finalize method and it requires at least two GC cycles to
deallocate,
> | so if no object is deallocated, from where the runtime gets the
space
> | to allocate the next object, will GC be fired again for these kind
of
> | cases when it does not able to release enough memory for the next
> | object or what.
> |
> |
> | Note:
> | The code works fine.No Exceptions so there is something missing in
my
> | understanding.
> |
> | Excuse me for such a long posting but i would highly appreciate if
> | someone clears my doubt where i am wrong.
> |
> | Thanks and Regards
> | Saurabh Garg
> |
> |
```

---

• *Follow-Ups:*

- ◆ **Re: How GC Works**
  - ◇ *From:* "Chris Lyon [MSFT]"

- **References:**

- ◆ **How GC Works**
  - ◇ *From:* saurabhgarg2k
- ◆ **RE: How GC Works**
  - ◇ *From:* "Chris Lyon [MSFT]"

- Prev by Date: **Re: MSIL**
- Next by Date: **Re: How GC Works**
- Previous by thread: **RE: How GC Works**
- Next by thread: **Re: How GC Works**
- Index(es):
  - ◆ **Date**
  - ◆ **Thread**