Analyse Memory Usage in iOS Apps

Ayush Gupta Follow
Jan 12, 2019 · 6 min read ★

Since 2007, iPhones have seen drastic changes whether its the structure or configuration. Memory(RAM) grew up from 128 MB to 4GB in 2018 whereas storage saw a rise from 4 GB to 256 GB. Yet memory remains an area of concern while developing for any smartphone application no matter which OS we are dealing with.

While working on an logging framework for iOS applications, multiple questions crossed my mind primarily based on memory management:

- How much memory my framework is using? Am I keeping too much of data in main memory?

- Are there any memory leaks?
- Will the client application crash with stress load due to high utilisation of RAM?
- How does garbage collection work in iOS? Is there a GC similar to what we have in java?

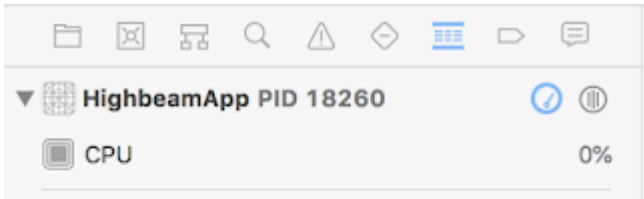
Let’s talk about each of the question and address them.

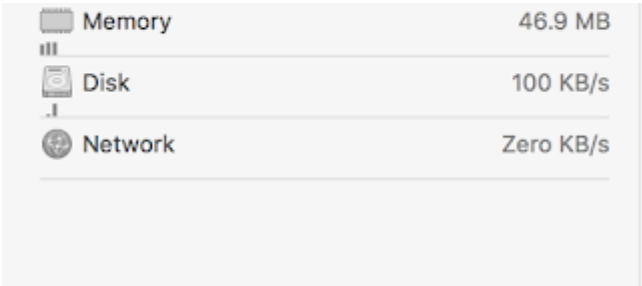
> How much memory my framework is using? Am I keeping too much of data in main memory?

Method 1

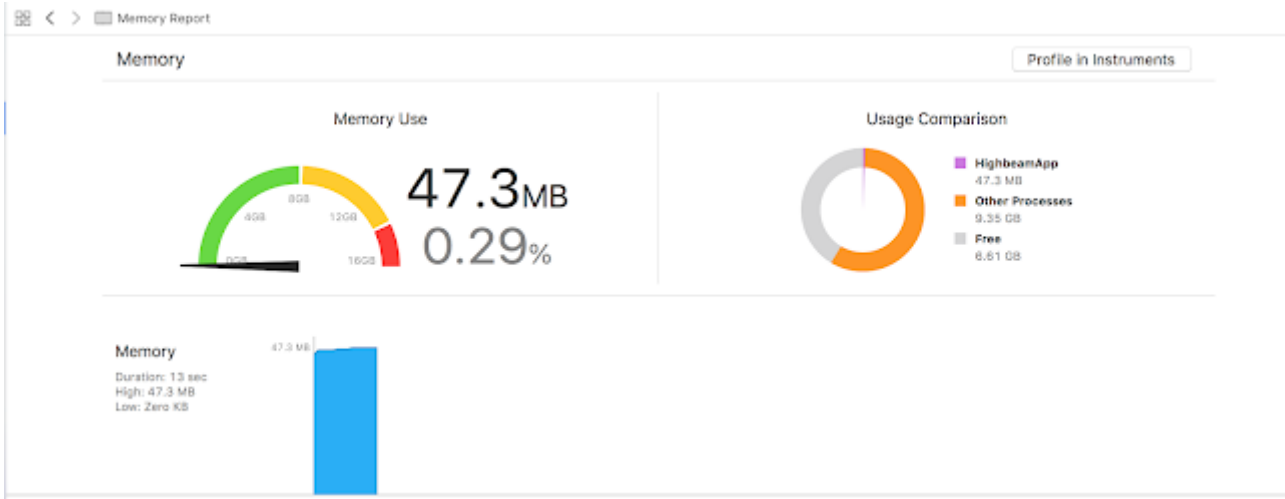
When we debug any iOS application, XCode provides usage metrics of CPU, memory, disk and network while in debug session.

To open Select the Debug Navigator (☷) option in the left pane





On clicking on each of the parameter, the main screen will show a detailed report. This is how memory report looks like:

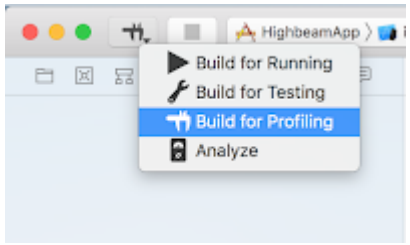


Although this report looks quite interesting and informative, it is rather misleading and lacks detailed breakage of memory. We will see that in later section.

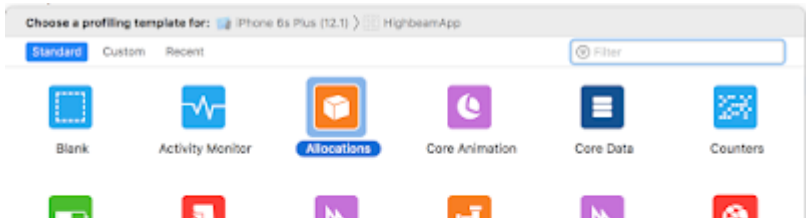
If you notice there is a small button on top right of this screen saying “Profile in Instruments”. Click on that. It will open Instruments app which shows a better and detailed view of how memory is being used by your application. We will explore Instruments app in detail later in the blog.

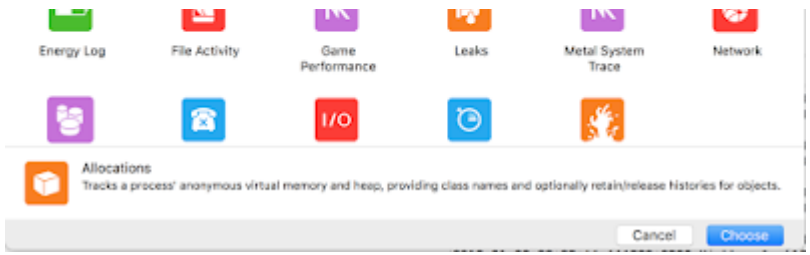
Method 2

We can also choose to build the application in **Profile** mode:



Instruments app will open asking about the way we need to profile our iOS application. There are numerous options to choose from based on the requirement.





For memory usage, we can either choose Allocations (shows memory usage in detail) or Leaks(show memory usage along with leaks)

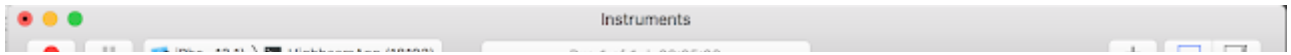
On the next screen, click on record button to start profiling and analysing usage.

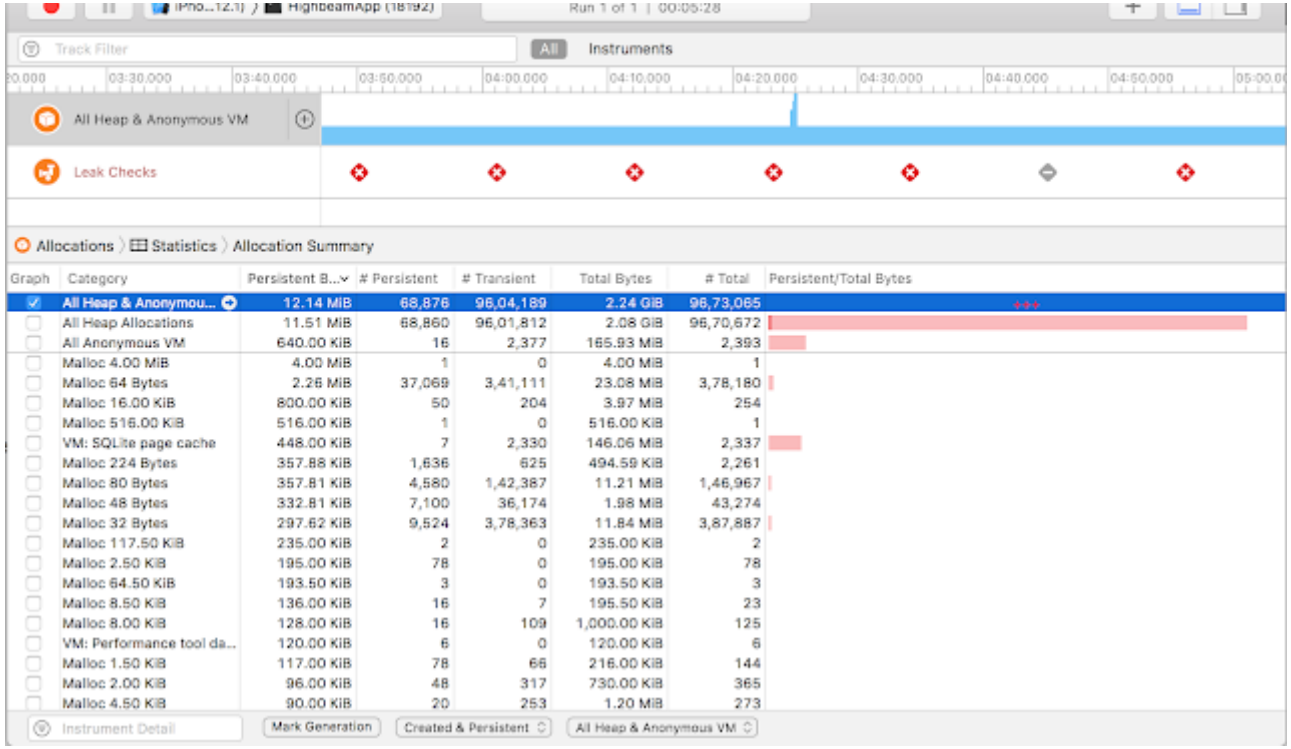
Instruments

As stated by wikipedia:

Instruments (formerly Xray) is an application performance analyzer and visualizer, integrated in Xcode 3.0 and later versions of Xcode.

We can view memory usage in Instruments using either of the two methods mentioned above. The screen will look like this:





Let’s try to understand major components of this screen:

1. **Graph** showing live memory usage on the top. By default combined memory usage of All Heap and anonymous VM is shown. You can choose to show additional graph by click on + sign right next to it.
2. In the table below there are multiple columns which are of great importance for us.
3. **Category** defines the where the memory is being used

4. **Persistent Bytes** defines the number of bytes currently held by the app in memory. This is of great use at it defines how much memory are we currently using.
5. **#Persistent** conveys number of persistent objects. These objects in total are consuming the bytes shown in previous column.
6. **#Transient** conveys the number of objects which were previously held by the app but are now release. They are no longer references by our applications. This might not be important while determining the memory usage but gives an overview of how the application used memory in the past.
7. **Total Bytes** is rather a misleading figures which combines the persistent and transient byes. This is only helpful if you want to know how much memory did your application used since the time its running.
8. **#Total** is summation of #Persistent and #Transient objects

Now as mentioned Xcode memory report is misleading. If we closely observe the memory usage value shown by Xcode and Instruments, they are quite different (one being 47.3MB and other 12.14MB). Also, I would suggest trying this out with an actual device than a simulator for exact results.

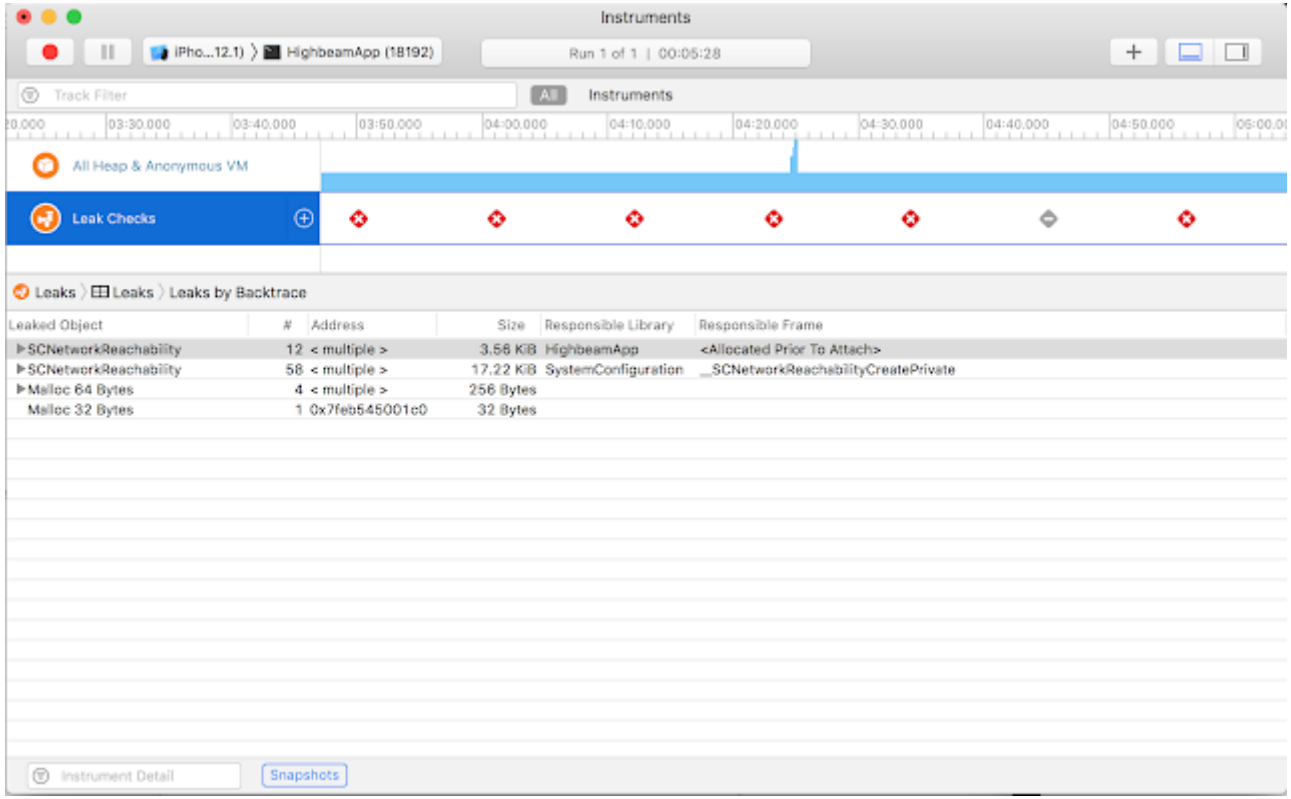
Now lets try understand the rows in the image shown above:

- The first row **All Heap and Anonymous VMs** is a sum total of all memory used by the app.
- The second and third rows are just a breakdown of first row into **All Heap Allocations** and **Anonymous VMs.** In short Row1 = Row2 + Row3
- All the rows below are granular breakdown of first row. In short Row 4 + Row N = Row 1.
  You can see two categories there:
- **malloc** — This memory is allocated in Heap. Summing up all of them would be equal to All Heal Allocations.
- **VM: xxxx** — This memory is allocated as Virtual Memory. Summing up all of them would give Anonymous VMs value.

Apple has provided a really nice documentation on Instruments under the title: Minimizing your app's Memory Footprint. It goes further deep into reducing memory footprint after detecting the issue.

> Are there any memory leaks?

As we have seen, Instruments app also help us in detecting in memory leaks in our application or integrated libraries. If we choose Leak Checks option in Instruments App, we will see something like this:



Here red crosses define memory leaks and we can even see the responsible Library in the table below. This is a great tool to catch the culprit.

> Will the client application crash with stress load due to high utilisation of RAM?

Apple is quite strict about memory usage by any application. If you exceed the memory usage limit, it will forcefully quit the application. Programmer generally has no control over this behaviour. The only solution is to reduce the usage for smooth functioning.

> How does garbage collection work in iOS? Is there a GC similar to what we have in java?

Garbage collection in iOS is different than in Java. In Java where we have a GC (garbage collector) dedicated to free up the memory for others to use. Unless GC completes its execution, the memory remains unusable.

In iOS, Automatic Reference Counting is used for garbage collection. For simplicity I'll be using the terms object and memory chunk separately although they both mean some object inside code.

- ARC keeps a count of how many objects are currently referring to a particular memory chunk.

- It increases/decreases the count as and when a object takes-up/leaves ownership of the memory chunk.
- When the reference count becomes zero, the memory is release by ARC.

In simple terms, if any object is referring to a memory chunk/object, it will not be released.

There are multiple scenarios related to strong and weak reference, which you can go through in this swift article: Automatic Reference Counting. This article has really good examples with proper explanation.

iOS Memory Management Memory Leak Profiling Xcode 8