

# Understanding memory usage in Docker Desktop on Mac

There have been many reports of possible memory leaks and apparent high memory usage of Docker Desktop since the upgrade to macOS 10.14 (Mojave). This document summarises the investigation into these reports and includes recommendations to help users better understand the memory usage of the system.

## Conclusions

1. The memory used by Docker Desktop does not grow over time, i.e., there is no memory leak.
2. The system is able to recover memory from Docker Desktop and give it to other processes if the system is under memory pressure. This is not reflected in the headline memory figure in the Activity Monitor, but can be seen by looking at the Real Memory.
3. The headline memory figure also has a double-counting bug in MacOS Mojave and later, causing it to report double the actual memory allocated. We have reported this bug to Apple.

## Recommendations

1. Add and then monitor the “Real Mem” column in Activity Monitor which counts exactly how much physical memory a process is taking right now. This value will move up and down depending on which part of the system needs the memory most.
2. Ignore the “Memory” column (the default column) in Activity Monitor which counts VM memory twice due to a bug in macOS 10.14 (Mojave) which has been reported to Apple.

## Understanding memory on the Mac

Modern operating systems like macOS use “virtual” memory. Each process has its own separate “address space” which can be much larger than the amount of physical memory in the machine. When a process allocates memory (e.g. via calling [malloc\(3\)](#)), macOS will simply mark a range of virtual addresses as used but it will not actually allocate any physical memory. Physical memory is allocated on-demand, when the virtual memory is first read or written. When some physical memory hasn’t been used for a while and the machine is under “memory pressure” (i.e. it is nearly full), macOS will save the contents of the memory to disk (a process known as “swapping”) so the physical memory can be recycled. If the process needs the memory later,

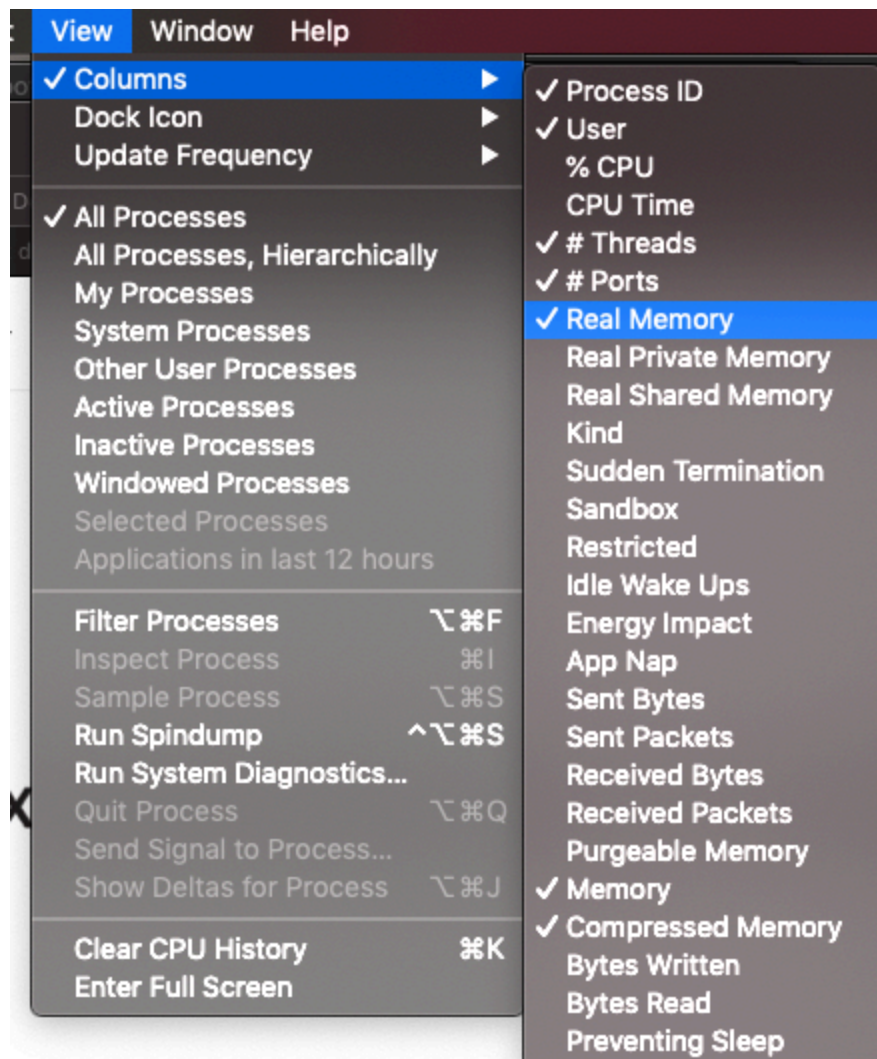
macOS will re-load it. More details of virtual memory on macOS are available in the [official documentation](#).

It is a common pattern for an application to “allocate” a large amount of memory and then only use parts of it at a time. The OS will only load memory that is actually needed into physical memory. Therefore to understand the true impact of a process on the system we need to watch how much memory the process is actually using, not the total it has “allocated”.

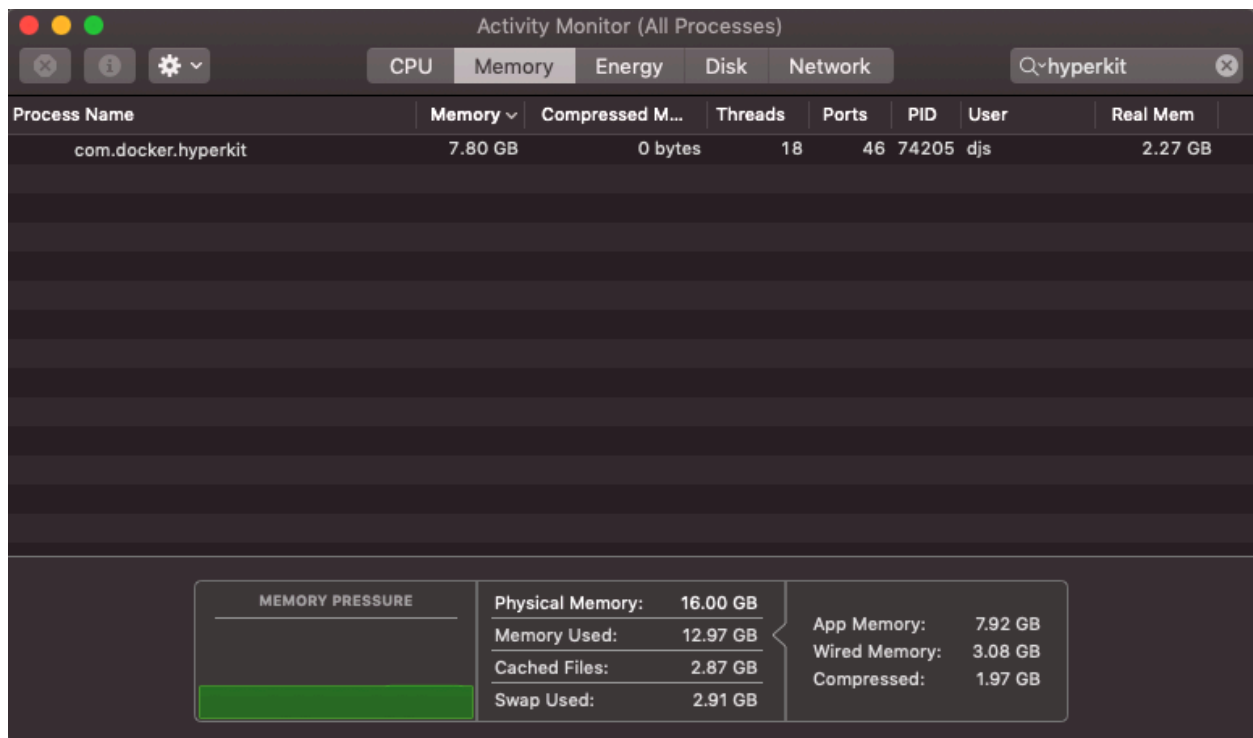
The following sections describe tools that we can use to understand the memory usage of processes on macOS.

## Activity Monitor

Activity Monitor is a GUI application which displays live CPU, memory, disk and network I/O information. On the “Memory” screen either double-click on the row to view the Real Memory or add the additional column “Real Memory” from the View / Columns menu:



Activity Monitor would then look like this:



The “Memory” column is a total of all memory which the application has allocated and written to at least once. Note that this never decreases unless an application officially deallocates memory by calling [free\(3\)](#), even if the memory is actually not still being used.

The “Real Mem” column (which we recommend adding) shows the current amount of physical memory the process is using at this moment in time. This will increase when the application uses more memory and decrease when macOS recycles it for some other purpose.

The “Memory Pressure” graph at the bottom of the window indicates how busy the memory system is. In the screenshot above the line on the graph is near the bottom and coloured green, indicating the system is not under pressure. Next to the graph are system-wide summaries of how memory is being used. Note that the system will try to use free memory for useful things like caching, so we expect most memory to be always be used.

## Command-line: ps

The command-line utility `ps` can be used to query memory information, for example:

```
$ ps -x -o pid,rss,vsz,command
PID    RSS      VSZ  COMMAND
328    3164   4305816 /usr/sbin/cfprefsd agent
329    5896   4334364 /usr/libexec/UserEventAgent (Aqua)
...
```

This displays the process id (PID), the Resident Set Size (RSS) (equivalent to “Real Mem” in Activity Monitor), Virtual Size (VSZ) (the size of the virtual address space of the process) and the command name.

Unfortunately there is no `ps` option which corresponds to the “Memory” column in Activity Monitor.

## Command-line: `vmmap`

The command-line utility `vmmap` displays and summarises the virtual memory allocated by a process, for example:

```
$ vmmap -summary com.docker.hyperkit
Process:          com.docker.hyperkit [74205]
Path:
/Applications/Docker.app/Contents/Resources/bin/com.docker.hyperkit
...

Physical footprint:          7.8G
Physical footprint (peak):  7.8G
...
```

The “Physical footprint” corresponds to the “Memory” column in Activity Monitor. This is new in macOS 10.14 (Mojave).

## Is there a memory leak?

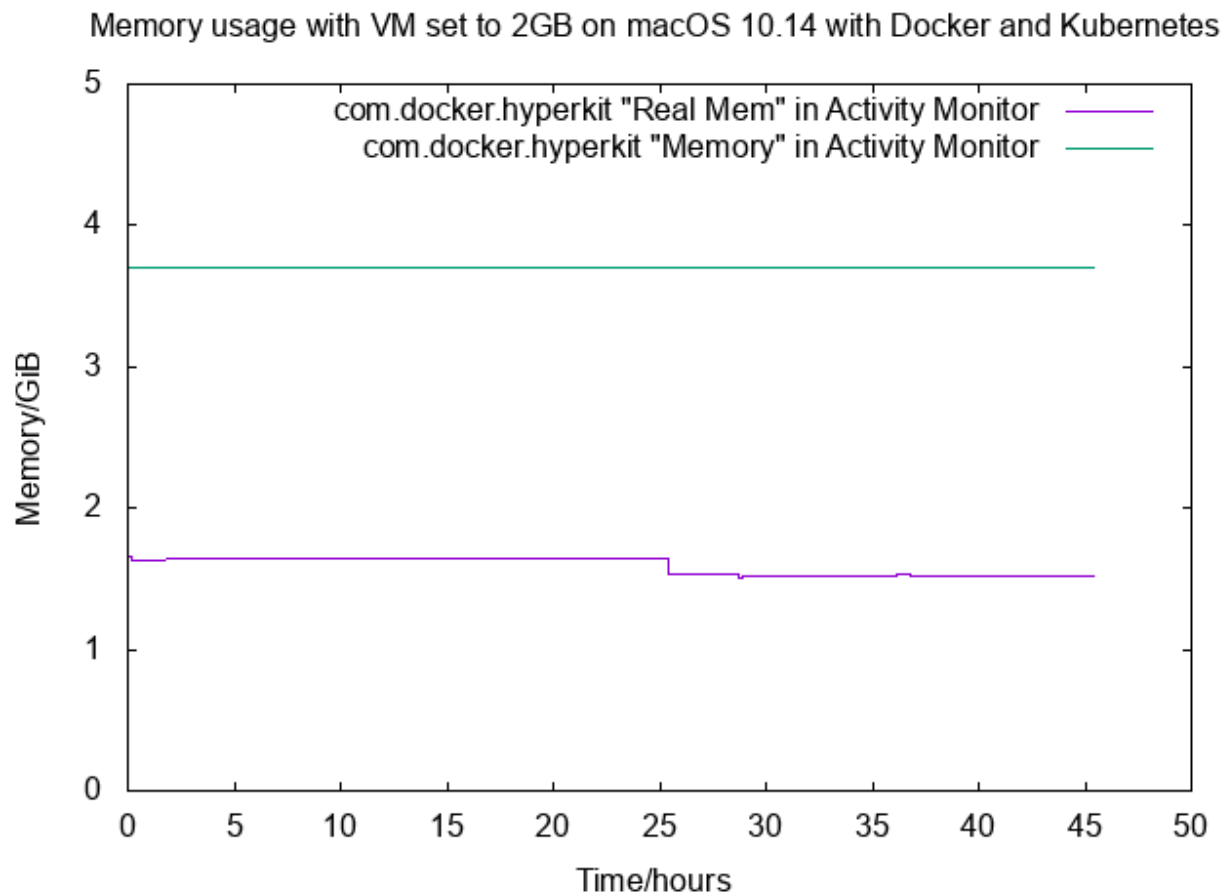
To discover whether there is a memory *leak*, we must monitor the memory usage over time. A leak would manifest as a steady increase in memory usage with no upper limit. Note that memory usage being high does not by itself demonstrate that a leak is present; we must show the increase over time.

## Experiment

We installed Docker Desktop on a MacBook Pro with macOS 10.14 (Mojave). We enabled Kubernetes and then monitored the “Memory” and “Real Mem” values using a combination of `ps` and `vmmap` using [this code](#).

## Results

The following graph shows the memory usage over an approximately 2 day long period:



Note that the reported “Memory” in Activity Monitor remains constant at just under 4GB but the “Real Mem” (which represents how much physical memory is actually being used) decreases a little over time.

## Conclusion

There is no sign of a memory leak while running an idle Docker and Kubernetes on Mac.

## What happens if there is memory pressure?

To understand what happens when the machine is running low on memory (under “memory pressure”) we must monitor Docker Desktop while running other memory-intensive applications.

## Experiment

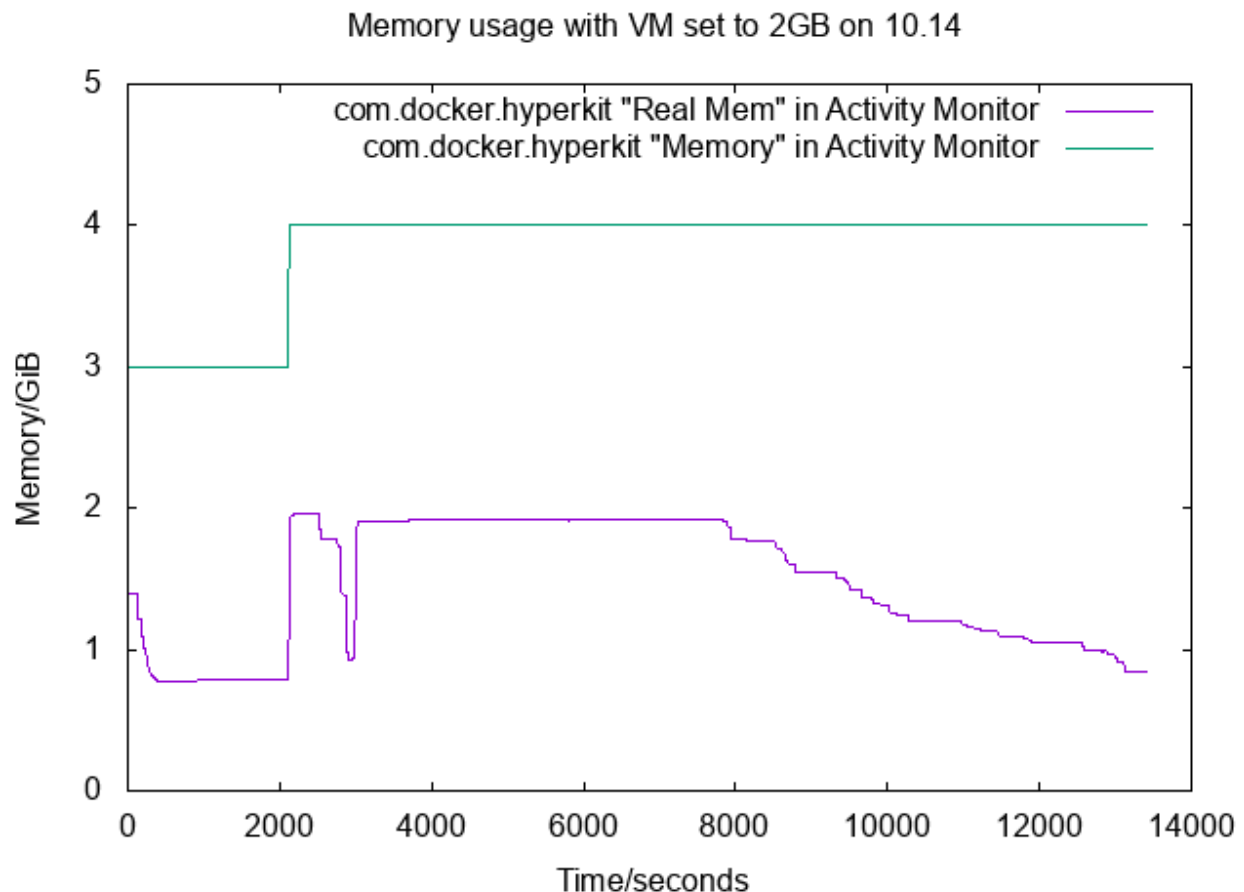
We installed Docker Desktop on a MacBook Pro with macOS 10.14 (Mojave). We then executed the following series of steps:

- For both 2GB and 4GB VM memory settings

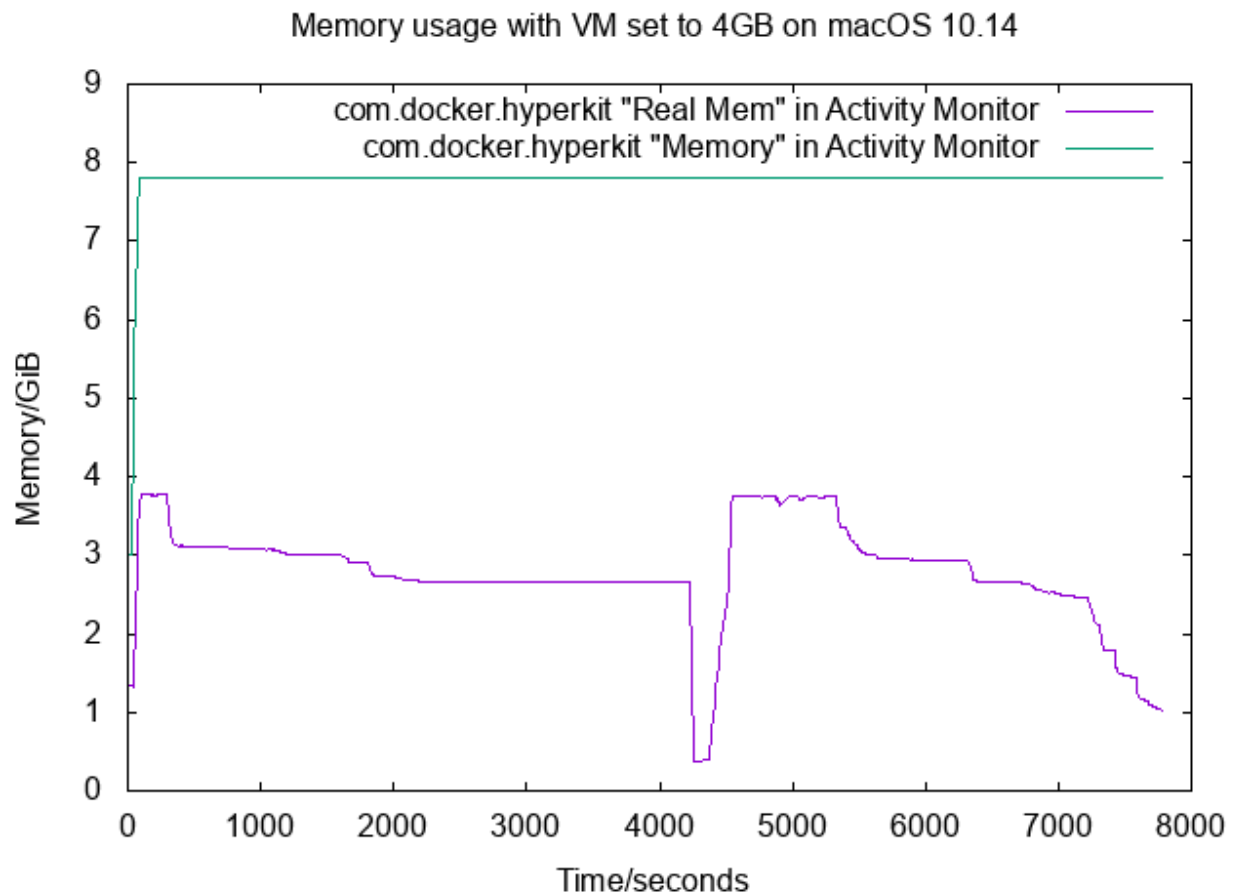
- Start Docker Desktop (with the default 1GB of swap)
- Repeat a few times:
  - Run a [container](#) which allocates and uses as much VM memory as possible (`docker run -it --privileged djs55/write-all-memory`)
  - Stop the container
  - Run [a program](#) which allocates and uses 15500 MiB on the host (which has 16GiB total), to cause memory pressure on the host
  - Stop the program

## Results

The following graph shows the memory usage of hyperkit when the VM is set to use 2GB in Docker Desktop settings:



Notice that the “Memory” in Activity Monitor rises to a maximum value (4GB) before remaining constant. The “Real Mem” (which represents how much physical memory is actually being used) goes up and down, depending on whether memory is needed inside the VM or on the host. The following graph shows the memory usage of hyperkit when the VM is set to use 4GB in Docker Desktop settings:



Notice that the “Memory” in Activity Monitor rises to a maximum value (8GB) before remaining constant. The “Real Mem” (which represents how much physical memory is actually being used) goes up and down, depending on whether memory is needed inside the VM or on the host.

## Conclusion

The “Real Mem” graph shows that macOS virtual memory system is able to move memory between the host and the VM, depending on where it is most needed. The graphs show that macOS can decrease the VM memory to the same level (approximately 1GB) irrespective of the VM memory setting in Docker Desktop preferences i.e. all possible VM memory is successfully freed and recycled.

## The misleading “Memory” column in Activity Monitor

The results from the earlier experiments demonstrate the “Real Mem” behaves as we would expect: when the VM needs more memory “Real Mem” goes up to the maximum given in Docker Desktop settings; when the host needs more memory it goes down. However the “Memory” in “Activity Monitor” is behaving strangely: “Memory” seems to always increase, up to

a maximum of 2x the memory size in Docker Desktop settings. To understand why this is happening we need to modify hyperkit itself.

## Experiment

The following patch was applied to hyperkit:

```
diff --git a/src/lib/vmm/vmm_mem.c b/src/lib/vmm/vmm_mem.c
index 2c5c858..eb1c8e7 100644
--- a/src/lib/vmm/vmm_mem.c
+++ b/src/lib/vmm/vmm_mem.c
@@ -47,11 +48,13 @@ vmm_mem_alloc(uint64_t gpa, size_t size)
     void *object;

     object = valloc(size);
     if (!object) {
         xhyve_abort("vmm_mem_alloc failed\n");
     }
-
+     sleep(30);
+     /* Write to the region from the host to cause all pages to be
    allocated */
+     memset(object, 0, size);
+     sleep(30);
     if (hv_vm_map(object, gpa, size,
                   HV_MEMORY_READ | HV_MEMORY_WRITE | HV_MEMORY_EXEC))
    {
```

The [vmm\\_mem\\_alloc function](#) is called to allocate memory for the VM. The memory is first allocated with [valloc\(3\)](#) on the host and then mapped into the VM using the function [hv\\_vm\\_map](#) in the hypervisor framework. The patch calls to [memset\(3\)](#) to write zeroes to the allocated region to make sure macOS allocates physical memory (recall physical memory is only allocated on-demand) and the calls to [sleep\(3\)](#) insert 30 second delays to make the effects of the allocation more obvious in Activity Monitor.

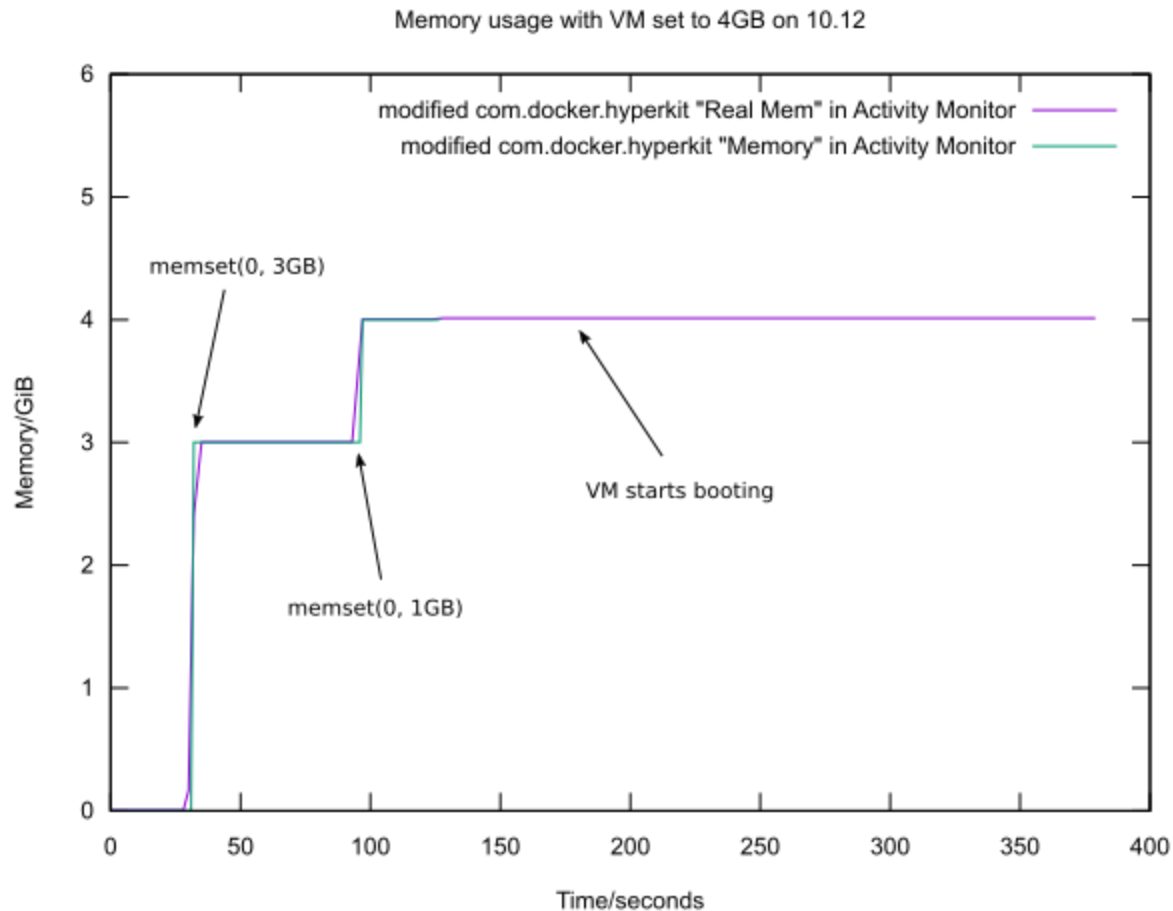
We executed the following series of steps:

- For both macOS 10.12 and 10.14
  - Start Docker Desktop with the modified hyperkit binary
  - Run a [container](#) which allocates and uses as much VM memory as possible (`docker run -it --privileged djs55/write-all-memory`).



## Results

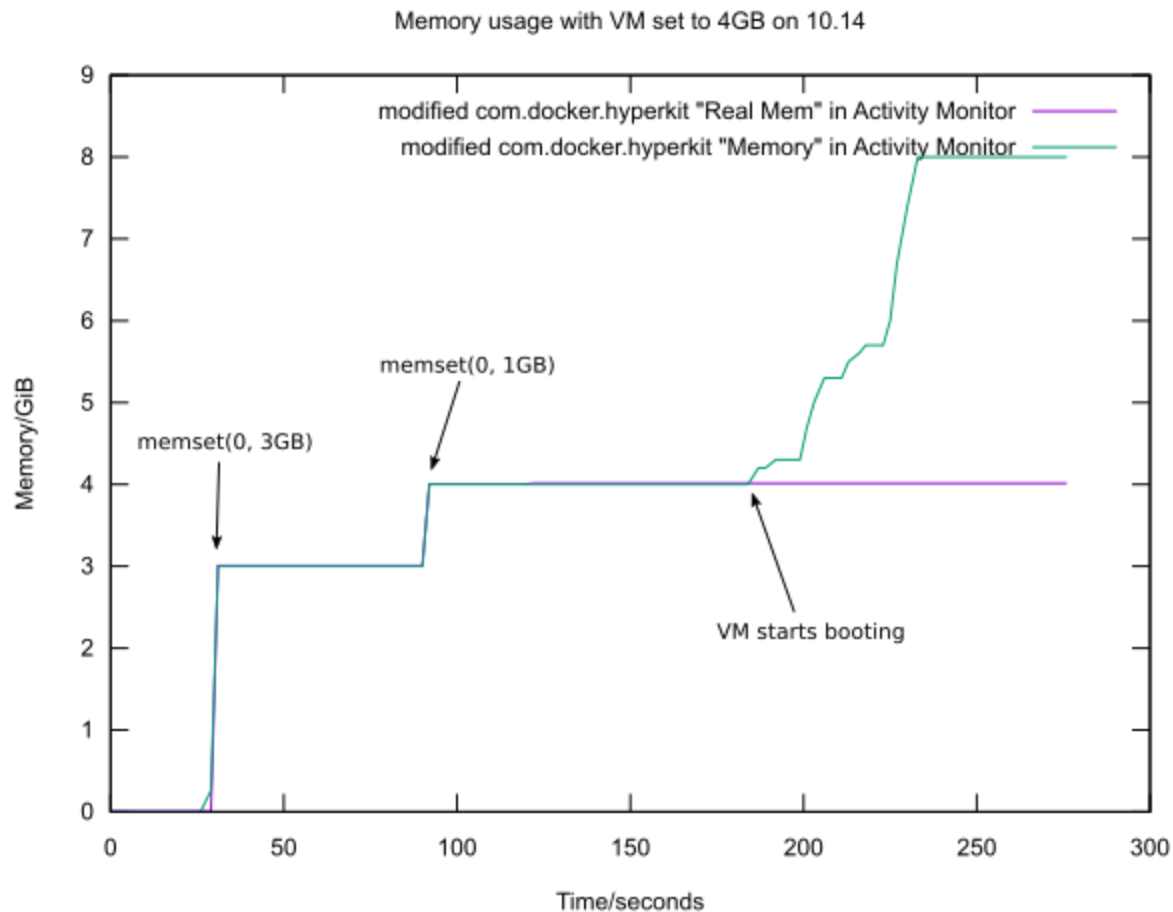
The following graph shows the “Memory” and “Real Mem” in Activity Monitor as the modified hyperkit starts, allocates memory and the VM boots on macOS 10.12 (Note the same behaviour is also seen on 10.13):



The 4 GB of VM memory is allocated in two chunks: the first is of size 3 GB and the second is of size 1 GB. When the memory is allocated with [valloc\(3\)](#) and written to with [memset\(3\)](#) both the “Memory” and “Real Mem” increase in Activity Monitor as expected.

When the VM starts booting and writing to the already-allocated memory, the “Memory” and “Real Mem” stay constant at 4 GB in Activity Monitor.

The following graph shows the “Memory” and “Real Mem” in Activity Monitor as the modified hyperkit starts, allocates memory and the VM boots on macOS 10.14 (Mojave):



As with macOS 10.12 and 10.13, we observe that the 4 GB of memory is allocated in two chunks: the first is of size 3 GB and the second is of 1 GB. When the memory is allocated with [valloc\(3\)](#) and written to with [memset\(3\)](#) both the “Memory” and “Real Mem” increase in Activity Monitor as expected.

However when the VM starts booting and starts writing to the already-allocated memory, the “Memory” in Activity monitor increases **again** but the “Real Mem” does not.

When the container in the VM writes to all the VM memory the “Memory” in Activity Monitor is 8 GB, 2x the 4 GB setting in Docker Desktop and 2x the “Memory” shown in macOS 10.12 and 10.13. “Real Mem” remains at 4 GB.

## Conclusion

It seems that Activity Monitor “Memory” is double-counting the VM memory in macOS 10.14. The memory is counted once when the hyperkit process allocates and writes to it and then counted a second time when the VM writes to it, even though it is the same memory. This

means that the “Memory” figure in Activity Monitor is often double the actual memory allocated. The “Real Mem” in Activity Monitor is behaving as expected.

## Locating the bug

The previous experiment shows that the “Memory” column in Activity Monitor appears to double-count memory shared with running VMs. Is the bug in hyperkit? Is hyperkit using the macOS virtualisation APIs incorrectly? Or is the bug in macOS itself? To determine whether the bug is in hyperkit or not, we can compare it to [gemu](#): another open-source hypervisor with a completely independent codebase.

## Experiment

We first created a small bootable test VM using [LinuxKit](#) including a small test container which detects and uses all VM memory. We built the VM image by:

```
$ git clone git://github.com/djs55/hyperkit-measure-memory
$ cd hyperkit-measure-memory/cmd/touch
$ make linuxkit-build
```

We then booted the VM with 2GB of memory using hyperkit on a MacBook Pro running macOS 10.14 (Mojave):

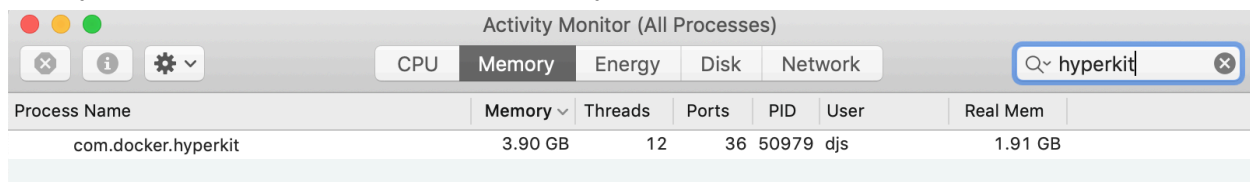
```
$ make linuxkit-run-hyperkit
```

We then booted the VM with 2GB of memory using qemu on the same MacBook Pro also running macOS 10.14 (Mojave):

```
$ make linuxkit-run-qemu
```

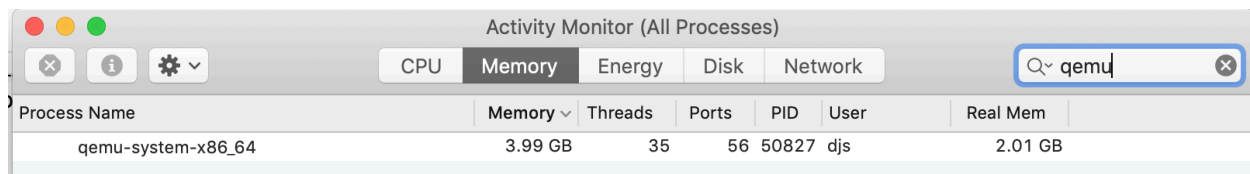
## Results

The hyperkit process looked like this in Activity Monitor:



Process Name	Memory	Threads	Ports	PID	User	Real Mem
com.docker.hyperkit	3.90 GB	12	36	50979	djs	1.91 GB

Note the hyperkit process is using just under 4 GB of “Memory” but around 2 GB of “Real Mem”. The qemu process looked like this in Activity Monitor:



Process Name	Memory	Threads	Ports	PID	User	Real Mem
qemu-system-x86_64	3.99 GB	35	56	50827	djs	2.01 GB

Note the qemu process is also using just under 4 GB of “Memory” and also around 2 GB of “Real Mem”.

## Conclusions

Since the qemu and hyperkit processes show the same double-accounting problem in Activity Monitor despite them being completely separate codebases, we conclude the bug must be in macOS itself, probably the Hypervisor.framework. We have reported the bug to Apple as bug number 48714544.