Android Application Usage Profiling

Profiling an Android Application

To gather application running statistics on the system, you can profile the system using Android Studio Performance Profiling tools https://developer.android.com/studio/profile/index.html.

The IDE provides a set of different tools which can effectively be used to profile any application running on a device or an emulator. To profile the memory usage, CPU/GPU usage or Network bandwidth usage, we can use the Android Monitor https://developer.android.com/studio/profile/android-monitor.html.

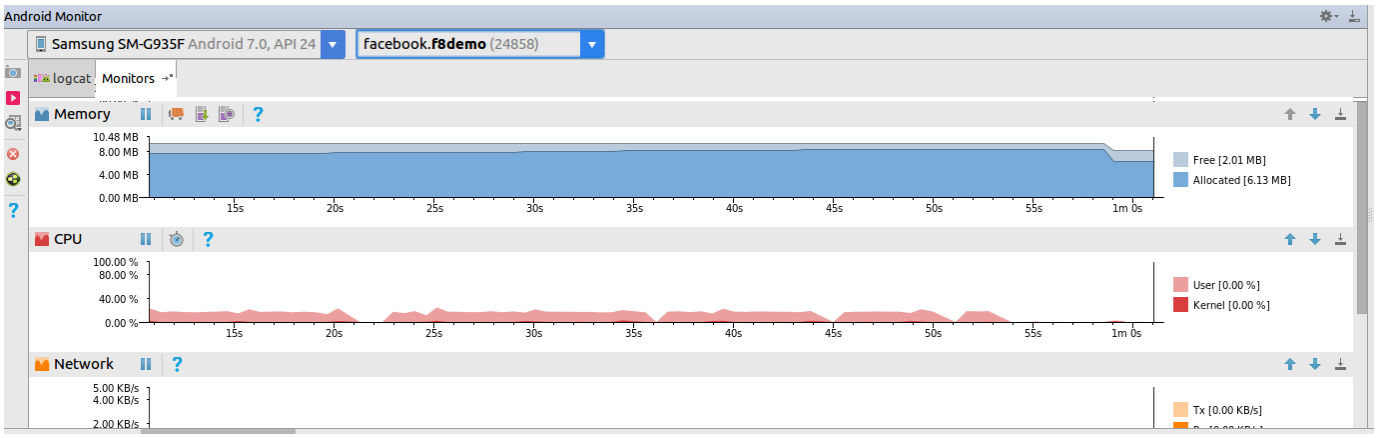
The Android Monitor helps us in these tasks:

- Log messages, which can be either system or hardware defined.
- Monitor memory, CPU and GPU usage by the application.
- Network bandwidth usage statistics.

To enable Android Monitor you need to take care of some prerequisites and dependencies as specified.

1. The device should remain connected to the system via USB cable and the system should be able to detect the device.
2. Enable ADB integration by selecting Tools > Android > Enable ADB Integration. Enable ADB Integration should have a check mark next to it in the menu to indicate it's enabled.
3. Make sure Android Device Monitor is not running currently.
4. In your app, set the `"debuggable"` property to `"true"` in the manifest or `"build.gradle"` file (it's initially set by default).

Now we just need to display the Android Monitor and when we run the application on the device, we can see its statistics be noted on the corresponding screen for "log messages (logcat)" or "performance monitors (monitors)". To view the Android Monitor, you can select it from View > Tool Windows > Android Monitor. Or you can also display Android Monitor by clicking on the Android Monitor button, which is on the bottom of the main window by default or using the shortcut Alt+6. A screenshot of an app being profiled is shown below.



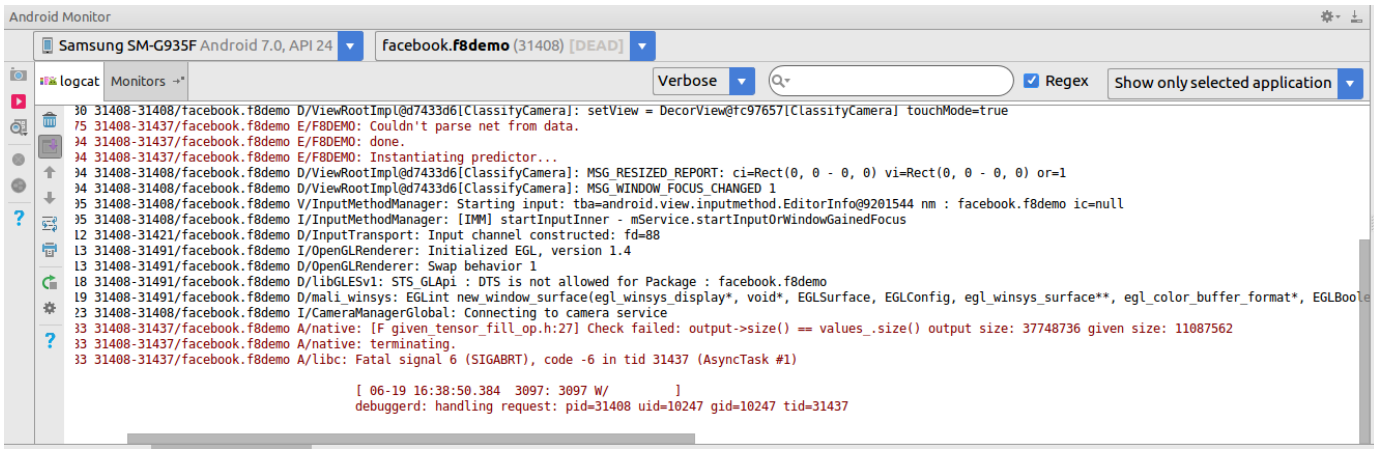
Gathering Memory Usage Statistics

Android Monitor provides us with various tools to check the usage statistics of the android application. Here we will list down some tools and try to understand how these help in understanding the memory usage of the application.

Log Messages

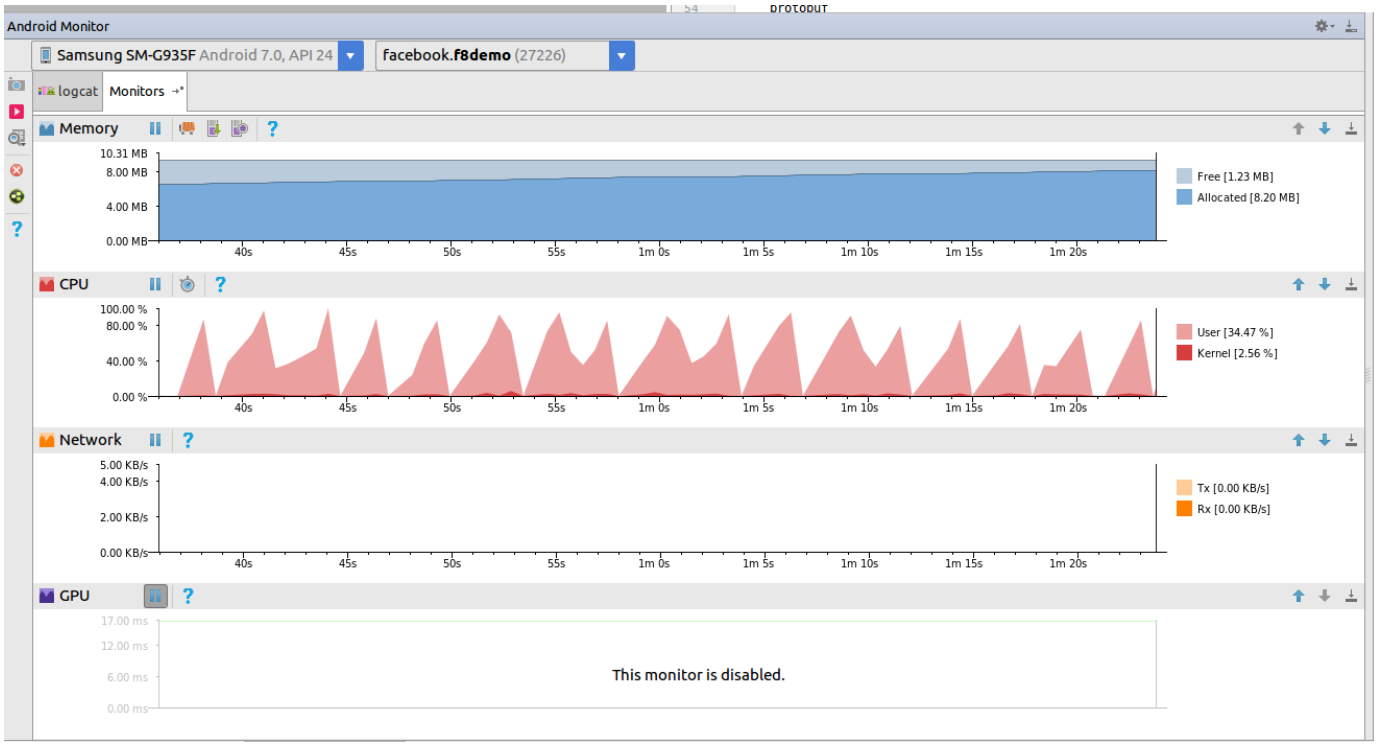
The "logcat" monitor gives us the log messages while the application is deployed and running in real time. It helps us in understanding what is

happening in the system which is very useful for debugging. For example if an application crashes the best way to find out the cause of the crash is to check the last few commands of the logcat. The screenshot below shows the logcat monitor while an app is running.



Performance Monitors

There are four types of performance monitors which helps us monitor the performance of the app in real time. We can get the statistics for Memory Usage, CPU Usage, Network Usage and GPU Usage with the help of this monitor. They give us tools to record specific additional data in files which can be analysed by using specific analysis tools. The screenshot below shows the performance monitor with each of the four resources being monitored in real time.



Data Analysis

System Information

In the Android Monitor main window, we can find the System Information icon on the side as marked in the screenshot below.

On clicking the icon you will get a list off system information that you can access. These are the various outputs of the "dumpsys" command.

- Activity Manager State - dumpsys activity
- Package Information - dumpsys package
- Memory Usage - dumpsys meminfo
    - A screenshot of the memory usage statistics is shown below. It gives us most of the information we need to know about the amount of memory being used by the application.

CMakeLists.txt × | android × | org.tensorflow.demo_2017.06.19_11.19.txt × | AndroidManifest.xml × | local.properties × | gradle-wrapper.properties ×

```
Applications Memory Usage (in Kilobytes):
Uptime: 236507865 Realtime: 978596080

** MEMINFO in pid 15067 [org.tensorflow.demo] **
                   Pss  Private  Private  SwapPss     Heap     Heap     Heap
                 Total    Dirty    Clean    Dirty     Size    Alloc     Free
                ------   ------   ------   ------   ------   ------   ------
  Native Heap   105435   105384       28       94   117760   100077    17682
  Dalvik Heap    65201    65168        0       82    17767    10661     7106
 Dalvik Other      817      816        0        0
        Stack      424      424        0        0
       Ashmem        4        4        0        0
    Other dev        5        0        4        0
     .so mmap    10250      388     7684       44
    .apk mmap      345        0        0        0
    .ttf mmap       54        0       24        0
    .dex mmap      564      500       64        0
    .oat mmap     2928        0      804        0
    .art mmap     1358      888       76        9
   Other mmap       18        4        0        0
   EGL mtrack    38304    38304        0        0
     Unknown      379      376        0       18
       TOTAL   226333   212256     8684      247   135527   110738    24788

 App Summary
                       Pss(KB)
                        ------
           Java Heap:    66132
         Native Heap:   105384
                Code:     9464
               Stack:      424
            Graphics:    38304
       Private Other:     1232
              System:     5393

               TOTAL:   226333       TOTAL SWAP PSS:      247

 Objects
               Views:        9         ViewRootImpl:        1
         AppContexts:        3           Activities:        1
              Assets:        3        AssetManagers:        2
       Local Binders:       11        Proxy Binders:       20
       Parcel memory:        5         Parcel count:       21
    Death Recipients:        2     OpenSSL Sockets:        0

 SQL
         MEMORY_USED:        0
  PAGECACHE_OVERFLOW:        0          MALLOC_SIZE:        0
```

- Memory Usage over Time - dumpsys procstats
- Graphics State - dumpsys gfxinfo

Some of the other Data Analysis tools which can be viewed using Android Monitor are as listed.

1. HPROF Viewer and Analyzer: This tools dumps the Java Heap to an HPROF file using the Memory Monitor. It's icon can be found, as one of the three icons, on the memory monitor just beside the play/pause button. The output of the dump can be viewed in the HPROF

Viewer which gives a nice class hierarchical view of the heap dump.
2. Allocation Tracker: This tool can also be found on the Memory Monitor which is used to track down all the memory allocation data in the app. It displays each method responsible for the allocation along with the size and the number of instances.
3. Method Trace
4. GPU Debugger