Abdul Manan
M.Abrar Tariq

# Midterm Project Report

## Browser Crash Analysis on Low Memory Mobile Device

**Objective:**

Analyze browsers on low end phones and figure out why do they crash and in what scenarios. More specifically, our goal was to study how memory management is done in android and what are its limitations that lead to a browser crash and how can we improve it.

**Milestones Achieved:**

1. Designed a native android app that takes Out of Memory Value (OOM Value) and amount of memory and create a memory pressure of given value while running at given priority (OOM_VALUE)
2. Deep analysis of android memory management, especially the working of LMKD and kswapd.
3. Measured Launch time of different browsers while maintaining different memory pressures at different priorities.
4. Analyzed the results of above experiment.

**Browsers Tested:**

- Chrome
- Firefox
- Microsoft Edge
- Yandex

**How our Native Application Works?**

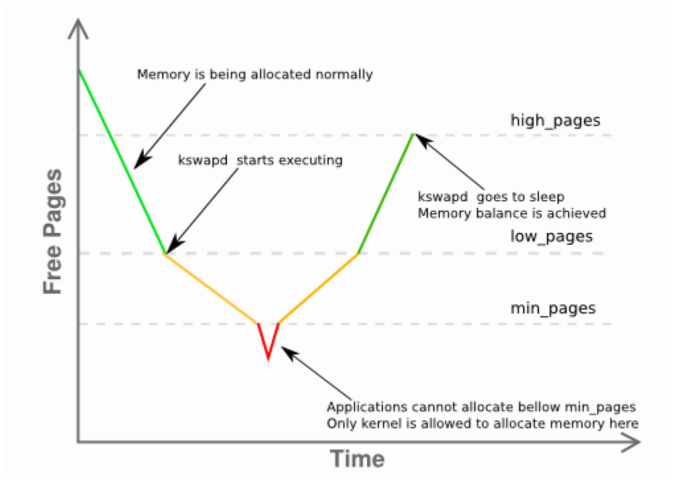
We have designed a native android app, **Daemon**, that generates a given memory pressure. It takes amount of memory in MBs to create memory pressure, priority of app to set, an app name to monitor and time that tells how long this pressure will retain. We have used these parameters because of following reasons:

- Memory Pressure itself is a subjective quantity. So, in our case, 15% memory pressure or 60% memory pressure that means 60% of RAM is currently being used by our app.
- Similarly, we have seen that app working on different priorities have different impacts on memory. For instance, an app having oom_value (priority) 17 means that will be killed first by kswapd. The table 1 below shows what different OOM values means.
- We have also added the functionality to monitor another app like Chrome because our purpose is to see how browsers behave in case of memory pressure and using this feature, we can see how browser react as we are increasing the memory pressure.

Our app does have some overhead, but we are measuring memory status with a gap of one second and just getting all results from one file */proc/meminfo*. This reduces CPU overhead to some extent.

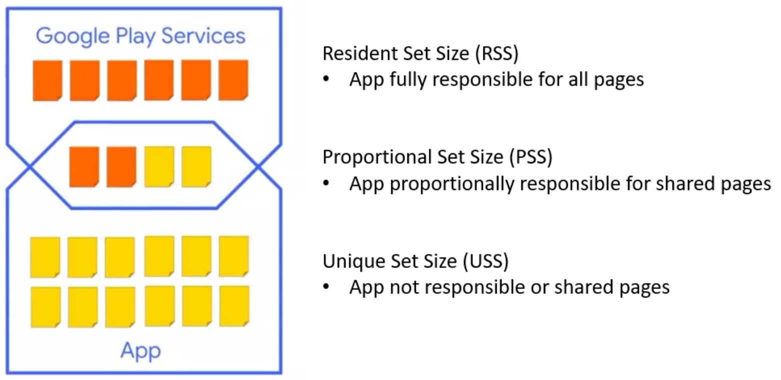
**How does LMKD and kswapd work?**

In Linux, certain memory threshold values are predefined, which represent the state of main memory. These Threshold values are known as watermark levels and are based on RAM size which is defined in file “mmzone.h” in Linux kernel 3.4. High watermark level is considered as safe available memory. Low and Min watermark levels are treated as memory crunch situation for system. As soon as the available memory goes below low watermark level, kswapd demon wakes up and starts freeing the pages. During kswapd execution, if still the allocation is aggressive and available memory has crossed Min watermark level, process asking for allocation, will itself go to Direct Reclaim (LMKD). Now, both LMKD and kswapd will free memory until it reaches the high watermark level. The below picture shows how kswapd and lmkd works as number of free pages changes.



**What Value to choose?**

Since android reports different types of values for memory, we have to decide each one's worth and follow the one that is best suited for our analysis. So, if you have an app who's calling into Google Play services, it's going to be sharing some memory with the Google Play services process. And then we can ask, how should we account for this shared memory? One approach is to use Resident Set Size (RSS). What this means is that when we're counting RSS, we're considering all the pages worth of memory that application is fully responsible for (Shared and Unique). Another approach is Proportional Set Size (PSS). In this case, we're going to say that the app is responsible for those pages which are proportional to the number of processes that are sharing them. Finally, there is a third approach called Unique Set Size (USS), where we say the application is only responsible for its unique pages. Now, in general, which approach to take really depends upon the context. The approach we took is the most straight forward one, which is proportional set size with equal sharing. One benefit of using PSS for evaluating an application's memory impact, especially when looking at multiple processes at the same time is that it will avoid overcounting or undercounting of shared pages.



Google Play Services
App
Resident Set Size (RSS)
• App fully responsible for all pages
Proportional Set Size (PSS)
• App proportionally responsible for shared pages
Unique Set Size (USS)
• App not responsible or shared pages

**Response Time:**

Response time is measured as how much time it took for launching an application after user request. It is measured using android logs so that we don't impose any extra overhead on android. We have taken onRequest and onDisplayed events from log file to measure response time. When we open a browser, it loads some files to see if user is connected to internet or not, make some network request(s) and may open user's last tab.

We have measured response time to see how these browsers behave in case of different memory pressure scenarios. For instance, in case of a memory pressure, there is a system call OnTrimMemory() that tells the app to release any resources like cache, our purpose is to see how these browsers behave to such events. The pictures below show how launch time is increased when we increase memory pressure from 20% to 40%. We have seen that Yandex performs poorly with an increase of 150% in launch time. Similarly, if we increase pressure to 50%, the performance drops drastically. One of the reasons is the app has already released cached or other extra resources and can not trim any more memory. Moreover, now CPU will also be used by kswapd and maybe lmkd resulting in an increase in launch time.

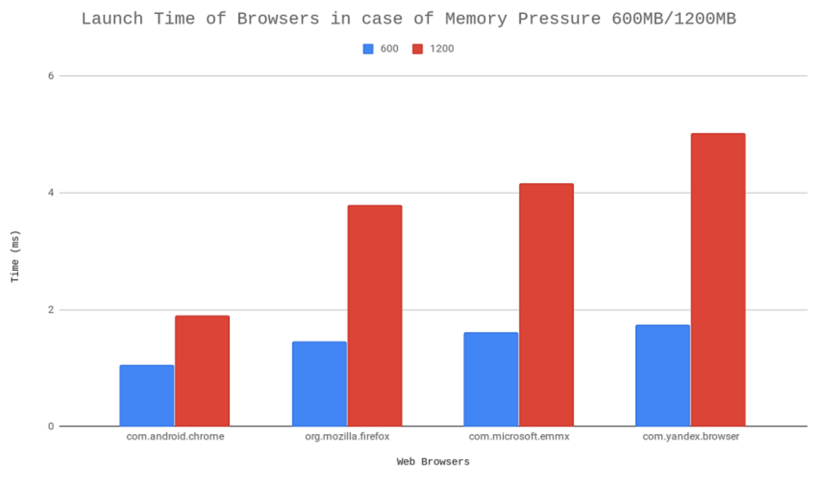


Image 1: Launch time of browsers in case of different memory pressures

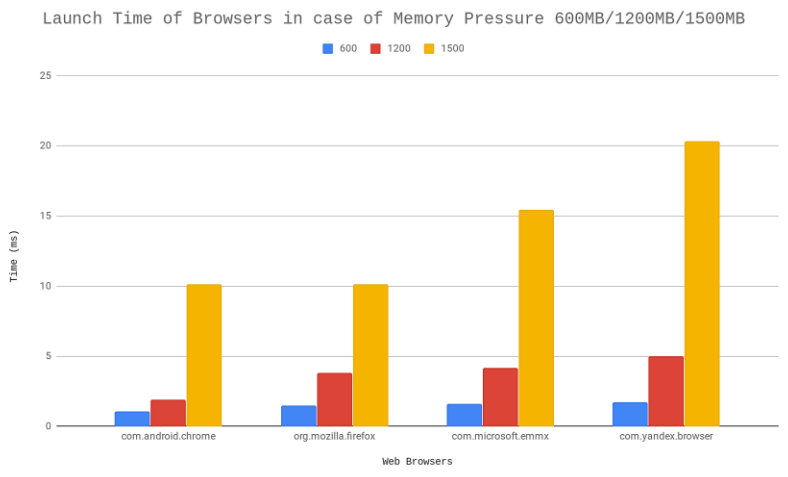


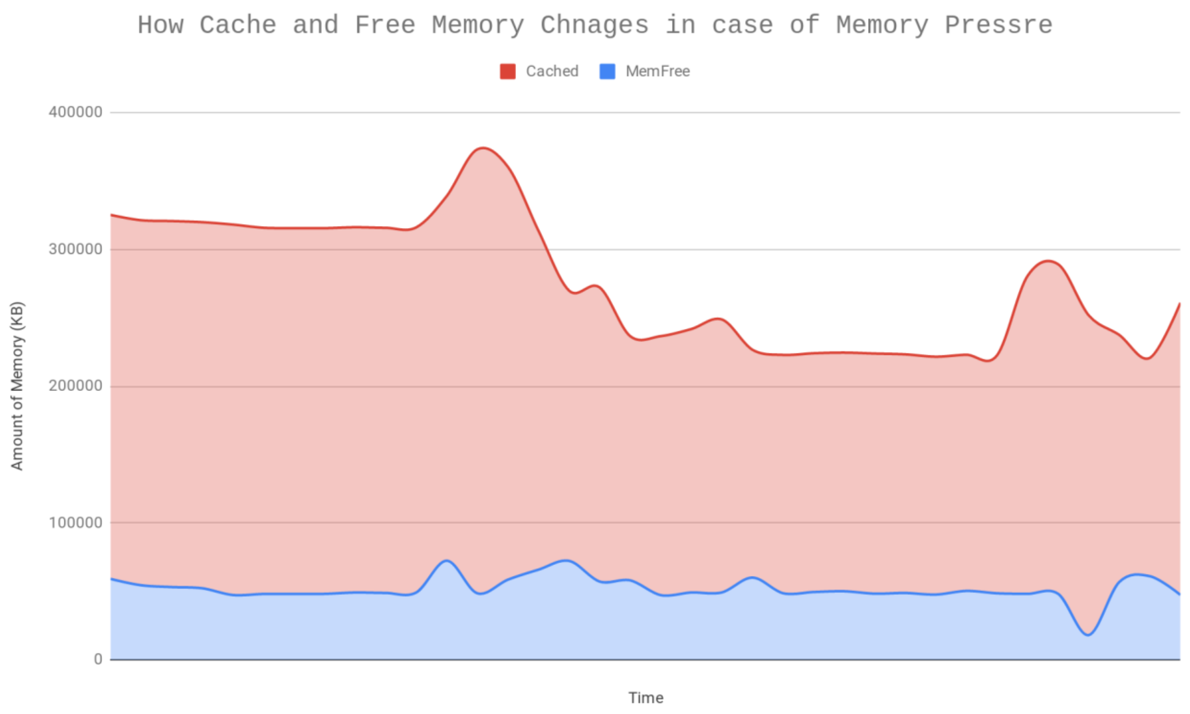
Image 2: Launch time of browsers in case of different memory pressures

One of the other observations is when a lot of memory is available, Chrome works better than other browsers. But, as we increase memory pressure, Firefox gives better performance. This is most likely because Firefox adjust itself on high memory pressure (OnTrimMemory) while others do not. Following table shows how PSS of these browsers change as we increase memory pressure. Note: Only browser homepage was opened.

| Browser | PSS (MB) | PSS (MB) + 50% Mem. Pres |
|---|---|---|
| Chrome | 130 | 122 |
| Firefox | 196 | 130 |
| Edge | 170 | 161 |
| Yandex | 188 | 189 |

**Other Observations:**

The below graph shows how free memory and cached memory changes as when we open Chrome browser.

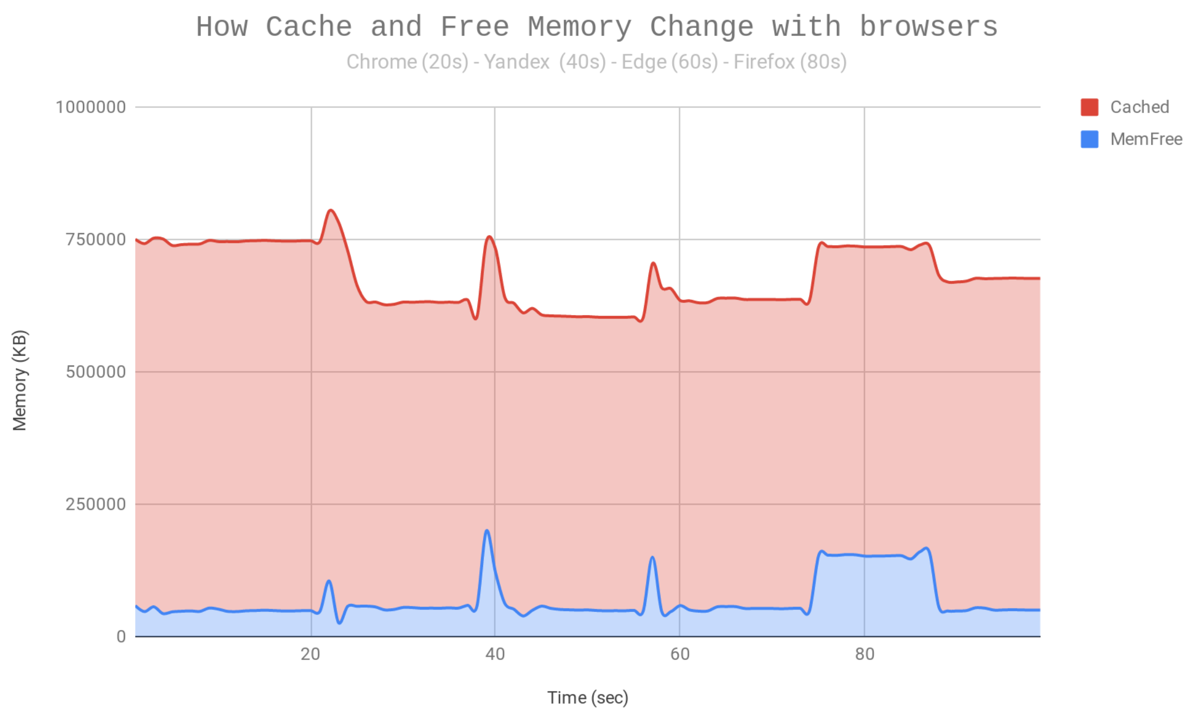


Some key observations are:

- Free memory remains ~50MB most of the time.
- Whenever free memory decreases, we see an increase in cached memory that is mainly because of kswapd.
- It also shows that lmkd and kswapd considers free memory mainly not cached memory.

The below graph shows how memory changes as we open a browser and then closes it. We have opened browsers in following sequence:

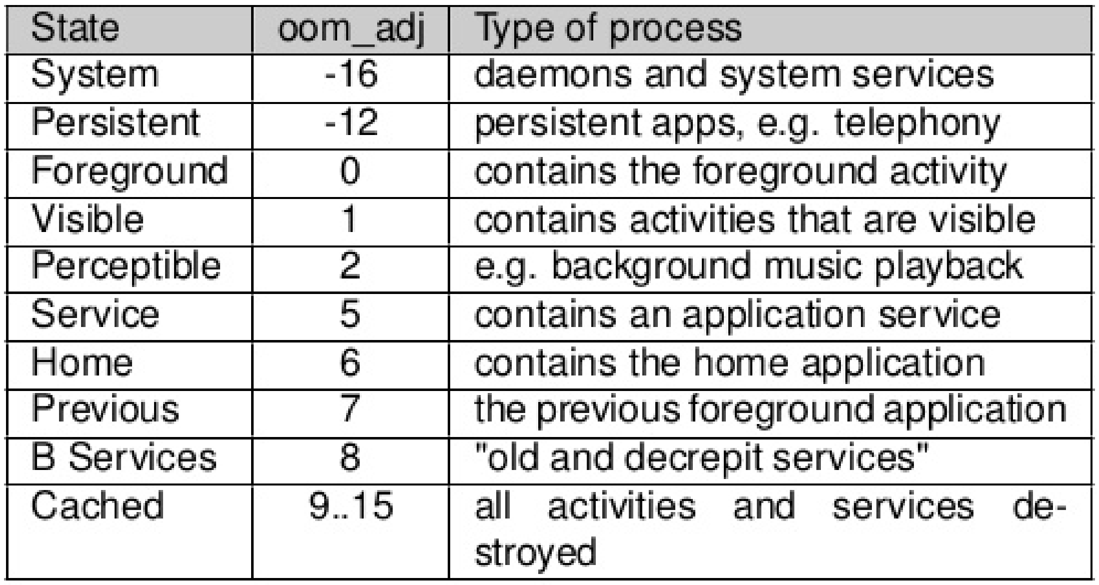
Chrome => Yandex => Edge => Firefox



The spikes shows how much memory got freed when we closed the browser. This graph shows Chrome takes less memory and as soon as we open Yandex, cached and free memory decreases and Yandex was the only browser that decreases free memory and some process got killed. Although Firefox has a large PSS value, but it is mainly shared memory and its USS (Unique Set Size) is less that makes it load faster than Yandex or Edge.

We have also seen that on memory pressure greater than 40% with priority negative 17, edge and Yandex mostly got crashed whenever we try to open a webpage. Firefox got crashed when memory pressure was 50% for some pages and for other pages, it works till 70% pressure. Moreover, Chrome remains opened till 60% of pressure and after that it started crashing. These results are not currently attached but will be provided in more detail in our final report.

Table 1: This table shows what different OOM_ADJ values (Out of Memory Adjust value)



| State | oom_adj | Type of process |
|---|---|---|
| System | -16 | daemons and system services |
| Persistent | -12 | persistent apps, e.g. telephony |
| Foreground | 0 | contains the foreground activity |
| Visible | 1 | contains activities that are visible |
| Perceptible | 2 | e.g. background music playback |
| Service | 5 | contains an application service |
| Home | 6 | contains the home application |
| Previous | 7 | the previous foreground application |
| B Services | 8 | "old and decrepit services" |
| Cached | 9..15 | all activities and services de-stroyed |

**Future Tasks to be carried out:**

1. Designed some synthetic web pages that takes a fixed amount of memory and record their PSS and other memory related items while maintaining different memory pressures at different priorities on 4 different web browsers.
2. Run same set of experiments on Nokia 1 having Android GO. We can see what memory optimizations are done in Android Go and Android 8 and how beneficial these optimizations are.
3. Android has made lmkd a lot more configurable in Android 7+ than before and we can see how see these settings affect the performance and how we can reduce crashes using these settings.

**Contribution of Each Member:**

Abdul Manan:

- Made Major part of NDK App for Generating Memory Pressure
- Added built-in support for reading memory status dynamically in app

M.Abrar Tariq:

- Made the Script for Parsing of android logs
- Added Support to detect process kill between Browser launch

P.S Creating of APP was a big Process and might requires a Document to explain

**Device:**

Following are the specifications of this device:

| **Device_1** | |
|---|---|
| **Test Location** | Pakistan (LUMS) |
| **Device** | Samsung Galaxy S6 |
| **CPU** | Octa-core (4x2.1 GHz Cortex-A57 & 4x1.5 GHz Cortex-A53) |
| **Internal** | 64 GB |
| **Ram** | 3 GB RAM |
| **Android** | 6.0.1 |
| **Browser** | Chrome |
| **Network** | Mobilink |
| **Internet Type** | Wifi |

**References**:

- http://admin.umt.edu.pk/Media/Site/icic/FileManager/Proceedings/Papers/42%20ICIC_2016_paper_42.pdf
- http://csl.snu.ac.kr/papers/tecs16.pdf
- https://www.ibm.com/developerworks/linux/library/l-adfly/index.html
- https://www.ibm.com/developerworks/linux/library/l-adfly/index.html
- https://jawadmanzoor.files.wordpress.com/2012/01/android-report1.pdf
- https://www.it.iitb.ac.in/frg/wiki/images/f/f4/113050076_Rajesh_Prodduturi_Stage-01_report_8_113050076_stage01.pdf
- https://s2group.cs.vu.nl/wp-content/uploads/2017/12/greenlab_final_report.pdf
- https://ieeexplore.ieee.org/document/7346725