Android Process Management FAQ

Notes on Android-specific aspects of Chrome process management

ppi@, August 2014

[What makes process management on Android different from Linux?](#)
[How does Android manage processes?](#)
[How does Chrome create renderers on Android?](#)
    [What is Crazy Linker a.k.a Chrome Dynamic Linker?](#)
[How does Chrome adjust the priority of its renderers on the system killing list?](#)
[How does Chrome know which process is in the foreground?](#)
[How is the number of spawned processes limited?](#)

What makes process management on Android different from Linux?

Creating child processes takes longer. On Linux, child processes are forked from the Chrome zygote process in a snap. On Android, Chrome needs to ask the operating system to create a process for us (forked from the **system** Zygote with Dalvik loaded there), and after that we still need to load our native library, separately for each renderer. All of this takes between 100 and 200 ms[1] on modern devices.

Once we have a child process, it's **hard to keep it around**. On Linux we have swap memory and the system is comfortable with keeping a lot of long-running processes. Processes can be killed by the out-of-memory killer but it is not a phenomenon to design / optimise for[2]. On

[1] Defined as the time between entering ChildProcessLauncher::LaunchInternal() and receiving a callback in ChildProcessLauncher::OnChildProcessLauncher. In particular, this includes the time needed to load Chrome native library.
[2] On Desktop Linux. On ChromeOS the oom killing shares the properties we have on Mobile.

Android there is no swap memory and the operating system regularly kills processes it considers “background”.

How does Android manage processes?

Being a mobile operating system, Android brings specific process management properties:

- **application processes are kept around** in background after the user exits / switches away from the app
- the OS **kills processes considered low-priority on demand**, whenever it runs out of memory

*Android keeps processes that are not hosting a foreground app component in a **least-recently used (LRU**) cache. For example, when the user first launches an app, a process is created for it, but when the user leaves the app, that process does not quit. The system keeps the process cached, so if the user later returns to the app, the process is reused for faster app switching.*

http://developer.android.com/training/articles/memory.html

The Android dumpsys tool allows to inspect the system list of processes ordered by their oom-killing priority. In case of memory pressure, the processes are killed from the bottom up:

```
fore  11569:com.google.android.apps.plus/u0a68 (top-activity)
prev  10880:com.google.android.gm/u0a31 (previous)
cch   11292:com.google.android.apps.maps/u0a30 (cch-act)
cch+1 11012:com.google.android.talk/u0a41 (cch-act)
cch+3 11167:com.google.android.keep/u0a85 (cch-act)
```

In this case the G+ app is in foreground, so the system is least likely to kill it. Google Keep app is the least recently used background app, so it will be killed first when the system will be short on memory.

How does Chrome create renderers on Android?

Android supports multi-process applications by allowing application components (e.g. services belonging to an app) to request **being run in separate processes** in the app’s AndroidManifest file. Chrome shell declares services for its renderers as follows:

```
<application android:name="org.chromium.chrome.shell.ChromeShellApplication">
    <activity android:name="org.chromium.chrome.shell.ChromeShellActivity" />
    <service android:name="org.chromium.content.app.SandboxedProcessService0"
             android:process=":sandboxed_process0" />
    <service android:name="org.chromium.content.app.SandboxedProcessService1"
             android:process=":sandboxed_process1" />
...
    <service android:name="org.chromium.content.app.SandboxedProcessService19"
             android:process=":sandboxed_process19"/ >
</application>
```

As the “android:process” properties of the renderer services are set to custom process names, the renderer services will be run in separate processes, different from the default main application process hosting the ChromeShellActivity activity.

Whenever we need to spawn a renderer, we request one of the declared services using bindService. This makes the system create a new process (as the service is declared as isolated) and load our service in it. Once ready, the service will execute its onCreate() method, this is where we load the Chromium native library in the newly created process.

What is Crazy Linker a.k.a Chrome Dynamic Linker?

Because Chrome renderers on Android are forked from the system Zygote and Chrome native library is loaded separately in each renderer, the RELRO section of the library is not shared between different processes, increasing the memory consumption by >1MB per renderer process.

To avoid this overhead, Chrome on Android employs a custom dynamic linker which loads the Chrome native library at the same address for all processes, mapping a single RELRO section backed by Android shared memory (ashmem) in each of them.

How does Chrome adjust the priority of its renderers on the system killing list?

A service bound to a foreground application by default is considered “visible” itself, and the system avoids it in out-of-memory killing.

Chrome cannot use the default behavior for the renderer services, as opening multiple tabs would saturate the system memory with high-priority processes. This would effectively kill all other background apps and, more importantly, services that were playing music or uploading files in background.

The Android framework allows to change the default behavior by specifying the relation between the main application process and the service processes using bind flags.

BIND_IMPORTANT	*this service is very important to the client, so should be brought to the foreground process level when the client is*
BIND_WAIVE_PRIORITY	*allows the service's process to be managed on the background LRU list just like a regular application process in the background*

http://developer.android.com/reference/android/content/Context.html

We use these flags to maintain multi-tab behavior without putting system stability at risk. All renderers are bound with BIND_WAIVE_PRIORITY flag through their entire lifetime, and get additional BIND_IMPORTANT binding while they're in foreground[3].

In this way the system is able to kill the background renderers like they were background apps when needed, while the foreground renderer remains at the top of the list. This translates to a system process list that may look as follows:

```
fore  14553:org.chromium.chrome.shell:sandboxed_process10/u0a17i31 (service)
fore  12241:org.chromium.chrome.shell/u0a17 (top-activity)
vis   3420:com.motorola.ccc/u0a0 (service)
vis   18436:android.process.acore/u0a1 (provider)
vis   27345:com.google.process.gapps/u0a32 (service)
vis   27357:com.google.process.location/u0a32 (service)
vis   27511:com.google.android.gms/u0a32 (service)
prcp  14087:com.android.inputmethod.latin/u0a44 (service)
svc   9492:android.process.media/u0a24 (started-services)
home  25886:com.android.launcher/u0a45 (home)
prev  13598:com.android.settings/1000 (previous)
cch   14268:com.android.calendar/u0a10 (cch-empty)
cch   14444:org.chromium.chrome.shell:sandboxed_process9/u0a17i30 (cch-client-act)
cch+1 14404:org.chromium.chrome.shell:sandboxed_process8/u0a17i29 (cch-client-act)
cch+2 15756:com.google.android.keep/u0a85 (cch-empty)
cch+3 13235:org.chromium.chrome.shell:sandboxed_process7/u0a17i28 (cch-client-act)
```

At the top of the list, we can see the browser process and the foreground renderer. Further towards the bottom, two background renderers are mixed with the background applications: Calendar and Keep. When the system runs short on memory, it will kill Chrome background renderer service 7 first and Keep shortly after; only after that the renderer services 8 and 9 will be killed.

How does Chrome know which process is in the foreground?

In order to correctly apply and lift the BIND_IMPORTANT binding as renderers enter and leave the foreground, Chrome needs to reliably distinguish between foreground and background renderers. Historically we had custom logic for that, now[4] we rely on the cross-platform ChildProcessLauncher::SetProcessBackgrounded call.

This is based on each RenderWidget keeping track of its visibility and notifying the respective RenderProcessHost about changes, which in turn signals the process as visible (foreground) whenever it hosts at least one visible widget.

---

[3] See BindingManager(Impl) in /content.
[4] http://crbug.com/325896

How is the number of spawned processes limited?

On Desktop the browser maintains a cap on the number of renderer processes[5] to be kept at once in order to limit the burden put on the operating system. The limit depends on the memory available on the machine, ranging from 12 renderers when there's only 1GB of RAM to the maximum of 82 renderers on computers with 16GB. If we reach the limit, new render widgets will be using one of the already existing processes, sharing the renderer process with other render widgets.

On Android, the operating system already takes care of killing our renderers when it can't sustain keeping them around. Therefore, Chrome doesn't limit the number of renderers being spawned on Android. This implies that on Android we never share renderers between unrelated tabs[6].

[5] actually it's a cap on the number of RenderProcessHosts, which is not the same thing as the host stays around when the process crashes

[6] http://crbug.com/325842