Published in ProAndroidDev

Ashu Tyagi Follow

Nov 13, 2020 · 4 min read · Listen

Save

Binders In Android (Part II): Reference Counting & Death Recipients

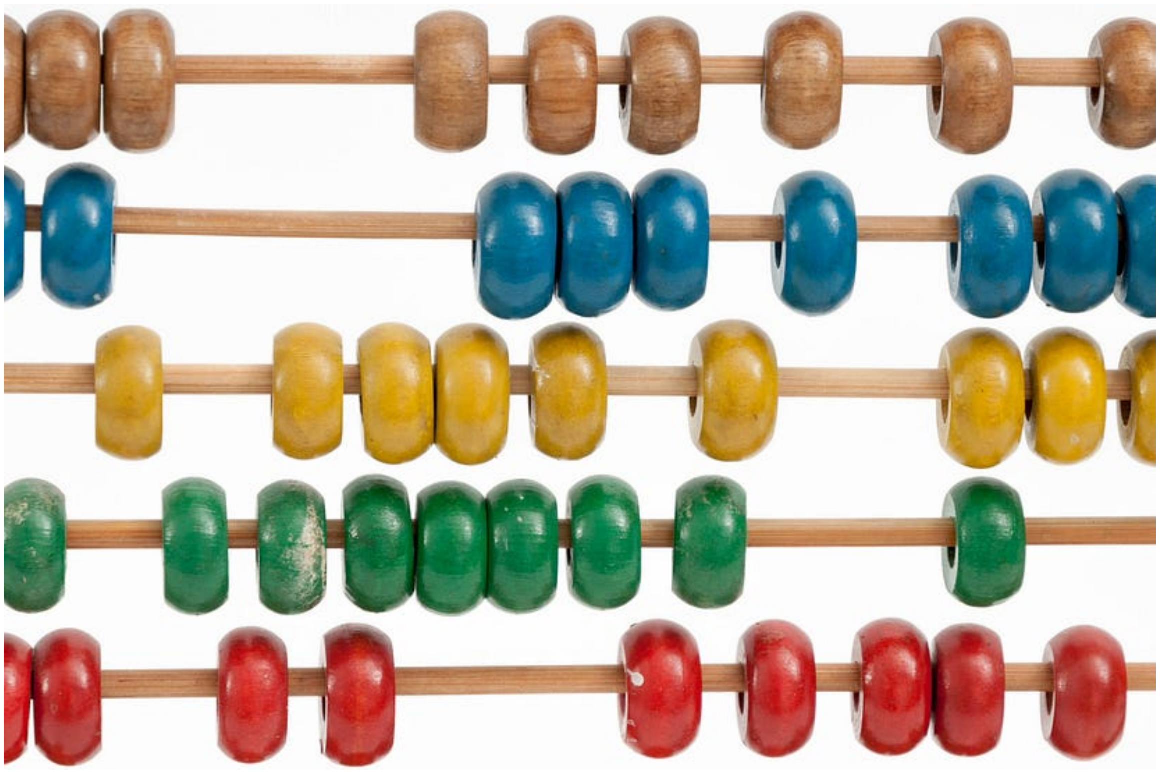


Photo by Crissy Jarvis on Unsplash

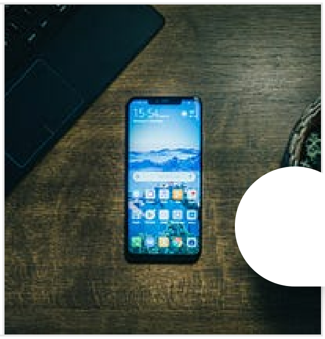
This is the second part of multi-part series about Binders in Android. It is highly recommended to read Part I (if you haven't already) otherwise this part probably won't make much sense.

Other articles in this series:

Binders In Android (part 1)

Introduction

proandroiddev.com



Introduction

In the first part, we saw that the binder is used for everything that happens across processes in the core platform. They are the driving force behind interaction in the framework and used extensively in the system for security. In this part, we'll talk about some very important features of binders which make it suitable for hostile environments. But why do we even need to know about death recipients & reference counting mechanism?

Since the very beginning, Android's central focus has been the ability to multitask. In order to achieve it, Android takes a unique approach by allowing multiple applications to run at the same time. Users usually never explicitly close the applications but instead, leave them running in the background. When memory is low, the Android system kills the low priority apps which are there in the background. This ability to keep processes waiting in the background speeds up app switching later down the line.

Android doesn't rely on applications being well-written and responsive to polite requests to exit. Rather, it brutally force-kills them without warning, allowing the kernel to immediately reclaim resources associated with the process. This helps prevent serious out of memory situations and gives Android total control over misbehaving apps that are negatively impacting the system. For this reason, there is no guarantee that any user-space code (such as an Activity's `onDestroy()` method) will ever be executed when an application's process goes away.

Considering the limited memory available in mobile environments

spread over dozens of system services (the Activity Manager, Window Manager, Power Manager, etc.) and several different processes. These system services need to be notified immediately when an application dies so that they can clean up their state and maintain an accurate snapshot of the system. Here enters the death recipient & reference counting. Now we have good enough reasons to know about these features, so let's dive into it.

Reference Counting

Reference counting guarantees that Binder objects are automatically freed when no one references them and is implemented in the kernel driver. The binder kernel driver does not know anything about binder objects that have been shared with a remote process instead what actually happens is:

1. Once a new binder object reference is found in a transaction, it is remembered by the binder kernel driver.

2. Every time that reference is shared with another process, its reference count is incremented.

3. The reference count is decremented either explicitly or automatically when the process dies.

4. When a reference is no longer needed its owner is notified that it can be released, and binder removes its mapping.

Death Recipient

Binder’s “link-to-death” to get a callback wh another process hosting a binder object goes away. In Android, any process can receive a notification when another process dies by taking the following steps:

1. First, the process creates a `DeathRecipient` callback object containing the code to be executed when the death notification arrives.

2. Next, it obtains a reference to a `Binder` object that lives in another process and calls its `linkToDeath(IBinder.DeathRecipient recipient, int flags)`, passing the `DeathRecipient` callback object as the first argument.

3. Finally, it waits for the process hosting the `Binder` object to die. When the Binder kernel driver detects that the process hosting the `Binder` is gone, it will notify the registered `DeathRecipient` callback object by calling its `binderDied()` method.

> Looking at the LocationManagerService source code helps us understand what’s happening under the hood (note that some of the source code has been cleaned up for the sake of brevity):

Let’s walk through the code step-by-step:

- When the application requests to get the location, the location manager service’s `requestLocationUpdates()` method is called. The location manager service registers the location listener for the application, and also links to the death of the location listener’s unique `Binder` token so that it can get notified if the application process is abruptly killed.

- When the application requests to remove location updates, the location manager service's `removeUpdates()` method is called. The location manager service removes the listener for the application, and also unlinks to the death of the location listener's unique `Binder` token (as it no longer cares about getting notified when the application process dies).
- When the application is abruptly killed before the location listener is explicitly removed, the Binder kernel driver notices that the location listener's Binder token has been linked to the death of application process. The Binder kernel driver quickly dispatches death notification to the registered death recipient's `binderDied()` method, which quickly removes the location listener and updates the device's location service.

Wrapping Up

The Binder's link-to-death & reference counting features are incredibly useful and are used extensively by the framework's system services. Many other system services depend on these facilities in order to ensure that expensive resources aren't leaked when an application process is forcefully killed. Some other examples are the `VibratorService`, `PowerManagerService`, `GpsLocationProvider`, and the `WifiService`.

Android | Android App Development | AndroidDev | Android Apps | Multitasking