**【VP】APP启动流程分析**

# 1 概览

## 1.1 源码下载

源码地址 <https://github.com/asLody/VirtualApp.git>

commit HASH: 17fc81c628594ecd2194cba774f1780c2047abbf

## 1.2 术语

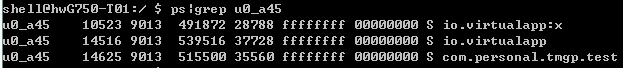
为描述方便，见一些常用的名字做一个约定。

* **AMS**: Acitivity Manager Service， Android系统实现的服务
* **VAMS**: Virtual Acitivty Manager Service，VirtualApp实现的Acitivity管理服务。
* **AMP**: Activity Manager Proxy（或者Client）在APP开发中有时候会用到Acitivity管理服务，实际上我们使用的是一个Proxy,通过Binder通信，让AMS实现真正的功能。
* **VAMP**:Virtual Activity Manager Proxy
* **PMS**: Package Manager Service
* **PMP**:Package Manager Proxy
* **VPMS**: Virtual Package Manager Service
* **VPMP**:Virtual Package Manager Proxy

## 1.3 VirtualApp vs Android

VirtualApp 能够创建一个虚拟空间让APP无感知地运行，与Android系统本身启动APP非常类似，这里做一个类比：

从VirtualApp里启动一个APP后，相关的进程列表如下：



* **VLauncher**: 宿主进程io.virtualapp,这个进程无特殊之处，非常类似Android系统的桌面，为描述便利，这里命名为VLauncher，便于与Android系统的Launcher进程比较。
* **VSystemServer**:服务进程io.virtualapp:x 这个进程由VLauncher通过BinderProvider启动，查看Manifest.xml文件，可以看到BinderProvier启动后，需要运行在另一个进程，与正常APP启动一个服务，要求这个服务运行在另一进程，没有太大区别。这个进程非常类似Android系统的system\_server进程，为描述便利，这里将其命名为VSystemServer
* **VApp**:从VirtualApp里启动的进程，为与通过Android系统的Launcher启动的APP做对比，这里将其命名为VApp。

# 2 分析

从VLauncher界面点击一个APP图标，完成一个APP的启动主要有以下几个步骤：

* **第1步**：从安装记录（在VirtualApp里安装一个APP的过程暂不分析）中获取对应App的相关信息，封装成一个Intent。主要包括APP的包名，入口Activity，APK路径等。主要函数是***Intent getLaunchIntent(String packageName,int userId)***。这个函数依赖VPMP来查询，最终在VSystemServer进程里的VPMS来实现。
* **第2步**：关键函数***int startActivity(Intent intent,int userId)***，通过调用VAMP里的这个函数来启动一个新的进程VAPP。经过层层调用，会执行到VAMS里的startActivity函数，这个函数运行在VSystemServer里，这与平常调用startActivity函数类似，真正的实现在system\_server进程里。
* **第3步**：关键函数 ProcessRecord performStartProcessLocked()，这个函数主要启动一个Provider。从闲置的StubContentProvider$CN（N可以是0到49）选择一个来启动，从配置文件AndroidManifest.xml可以看出，启动的Provider需要运行在一个新进程中，新进程的名字是io.virtualapp:pn（n可以是0到49），这个过程与正常的APP启动一个Provider，并且要求这个Provider在新进程中运行，没有任何区别。**可以看出VirtualApp启动一个Provider的任务交给Android系统来完成**。
* **第4步**：既然启动了一个新进程，自然要运行VApp.java里的代码，在attachBaseContext运行完成后，就完成了VPP进程自身的大部分HOOK，如AMS,PMS等。VAPP替换了很多系统服务，未研究。**这个步骤的代码是在VAPP里运行的**。
* **第5步**：启动StubActivity$CN（或者StubDialog$CN，根据VAPP的配置来选择），调用系统的startActivity函数来启动一个Activity,**在这里VirtualApp仍然是将启动Activity的任务交给了系统**。这个步骤是运行在VSystemServer进程里的。有AndroidManifest.xml配置这个Activity需要运行在一个新进程，就是刚才的io.virtualapp:pn进程里，其实这个进程已经启动，且相关函数已经被HOOK。
* **第6步**：handleLaunchActivity这个步骤运行在VAPP里，系统在启动一个Activity时，会向运行这个Activity的进程发送LAUNCH\_ACTIVITY的消息，处理这个消息的是ActivityThread里的回调函数H，由于新进程VPP在启动时，已经HOOK了对应的函数，这个消息我们可以自己处理后，再还给系统的默认处理函数。我们接管这个消息做了两件事情。就是下面的第7步和第8步。
* **第7步**：bindApplication,主要完成修改进程名为包名，如果有Application就启动Application,并调用onCreate函数，安装异常处理函数等。
* **第8步**：将stubActivity替换成真正需要启动的Activity。完成VAPP的启动。

从上面的步骤可以看出，VSystemServer模拟了部分system\_server进程的功能，主要用于管理从VLauncher启动的VApp。（从AndroidManifest文件可以确定，最多可以启动50个VAPP）VSystemServer进程需要知道这些VAPP的运行状态，如果VAPP退出，VSystemServer需要将这个VAPP占用的相关资源回收（如vpid,ActivityRecord等）。

启动的流程图：

