Introdução a Arquitetura Android

Maycon Viana Bordin

[1]Bacharelando em Sistemas de Informação – Sociedade Educacional Três de Maio
Caixa Postal 153 – CEP 98.910-000 – Três de Maio – RS – Brasil

mayconbordin@gmail.com

Resumo. *Sendo o sistema operacional com maior presença entre os smartphones em todo o mundo, desenvolver aplicações para o Android tem se tornado cada vez mais interessante. Entretanto, para tirar o melhor proveito desta plataforma é importante conhecer como ela funciona internamente. Este artigo aborda os principais componentes que fazem parte do* software stack *do Android, começando pelo kernel Linux e a Dalvik VM até os componentes principais de uma aplicação. E mostra as estratégias adotadas pela plataforma para lidar com características inerentes aos dispositivos móveis, como o tempo de bateria e a baixa capacidade de memória.*

Abstract. *As the operating system with the largest presence among smartphones worldwide, developing applications for Android has become increasingly interesting. However, to take advantage of this platform it is important to know how it works internally. This paper discusses the main components that are part of the Android software stack, starting with the Linux kernel and the Dalvik VM to the main components of an application. And shows the strategies adopted by the platform to deal with inherent characteristics of mobile devices such as the battery life and low memory capacity.*

1. Introdução

O Android é um sistema operacional para dispositivos móveis *open source* baseado em Linux e desenvolvido pela Open Handset Alliance, liderada pelo Google [Brähler 2010].

No Android, aplicações são desenvolvidas em linguagem de programação Java, utilizando o Android SDK, e executam na máquina virtual Dalvik. O Android, entretanto, possui suporte para o desenvolvimento de aplicações nativas, escritas em C e C++, através do Android NDK [Brähler 2010].

Aplicativos escritos para Android são distribuídos através da loja de aplicativos do Google Play, antigamente conhecida como Android Market, e que conta atualmente com mais 500.000 aplicativos, tanto pagos como gratuitos, e que já foram baixados mais de 15 bilhões de vezes [Lunden 2012, Google 2012].

O sistema operacional também está presente em mais de 56% dos smartphones em todo o mundo, tornando o Android uma das plataformas mais interessantes para o desenvolvimento de aplicativos [Gartner 2012]. E para tanto, é importante conhecer como este sistema operacional que vem ganhando tanta popularidade funciona, para a criação de aplicativos que façam bom uso dos recursos disponíveis.

Nas próximas seções deste artigo serão analisados em detalhes o funcionamento dos principais componentes da arquitetura do Android.

2. Arquitetura

Na realidade, como é colocado em [Android Developers 2012], o Android é mais do que um sistema operacional. Ele é na verdade um *software stack* composto por cinco camadas, como mostra a Figura 1. A base do Android é uma versão modificada do kernel Linux 2.6, que provê vários serviços essenciais, como segurança, rede e gerenciamento de memória e processos, além de uma cadamada de abstração de hardware para as outras camadas de software.

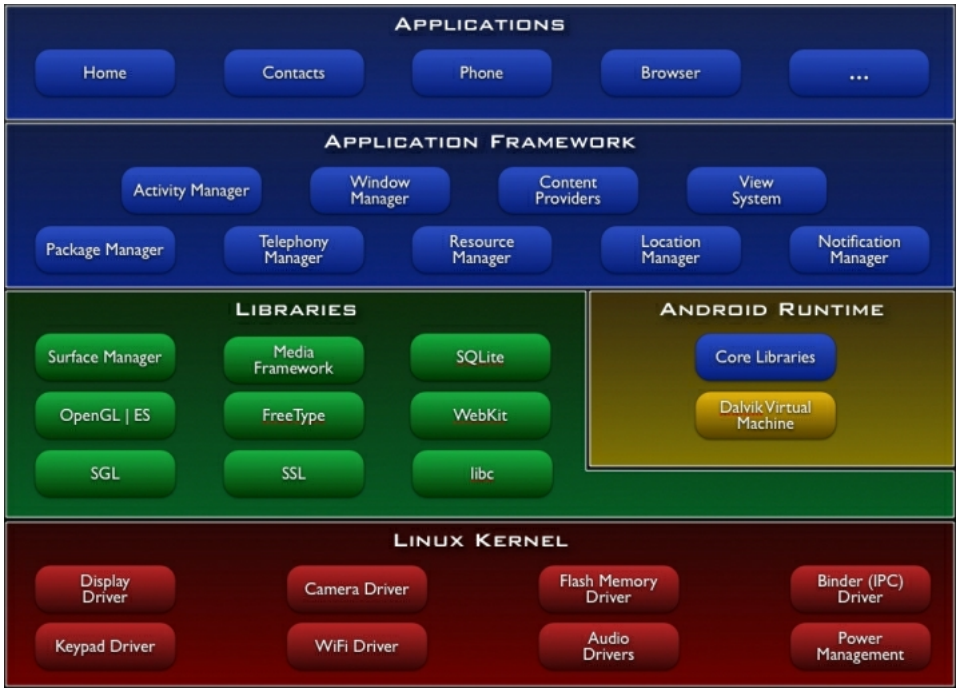

Figura 1. Arquitetura Android.

Acima do kernel ficam as bibliotecas C/C++ utilizadas por diversos dos componentes do sistema, como: uma implementação da biblioteca padrão do C (libc), mas com licença BSD e otimizada para dispositivos embarcados; bibliotecas para suporte a formatos de áudio, vídeo e imagens; um gerenciador que intermedia o acesso ao display e compõe as camadas de imagem 2D e 3D; o engine para navegadores WebKit; bibliotecas para gráficos 2D (SGL) e 3D (OpenGL ES); um renderizador de fontes bitmap e vetoriais; e o banco de dados relacional SQLite [Android Developers 2012].

No Android, aplicações escritas em Java são executadas em sua própria máquina virtual, que por sua vez é executada em seu próprio processo no Linux, isolando-a de outras aplicações e facilitando o controle de recursos [Android Developers 2012]. O Android Runtime é composto pela máquina virtual chamada Dalvik VM, onde as aplicações são executadas, e por um conjunto de bibliotecas que fornecem boa parte das funcionalidades encontradas nas bibliotecas padrão do Java [Android Developers 2012].

Na camada acima, escrita em Java, fica a framework de aplicações, que fornece todas as funcionalidades necessárias para a construção de aplicativos, através das bibliotecas nativas. Aplicações Android podem possuir quatro tipos de componentes: *activities, services, content providers* e *broadcast receivers*. Além destas peças fundamentais em

uma aplicação, existem os recursos, que são compostos por layouts, strings, estilos e imagens e o arquivo de manifesto, que declara os componentes da aplicação e os recursos do dispositivo que ela irá utilizar [Android Developers 2012].

3. Kernel Linux

Como fora colocado na seção anterior, o Android utiliza o kernel Linux para prover vários serviços as aplicações que nele são executadas. Entretanto, esta versão do kernel Linux foi modificada, de modo a melhor atender as necessidades existentes em dispositivos móveis, como as de gerenciamento de memória e energia e do ambiente de execução [Brähler 2010].

Em [McDermott 2008] é descrito o processo de portabilidade do Android 1.0 para um dispositivo Nokia, evidenciando que o kernel do Android contém 75 arquivos modificados, além de 88 arquivos adicionais com relação ao kernel original. Dentre as modificações estão: uma CPU virtual, chamada Goldfish, que executa instruções ARM926T, e que é utilizada pelo emulador Android; o sistema de arquivos de alto desempenho para memórias flash YAFFS2 (*Yet Another Flash File System, 2nd edition*); correção de bugs relacionados ao Bluetooth, além da adição de funções para controle de acesso e debug.

O kernel utilizado pelo Android também introduz o Binder, um novo mecanismo para a comunicação entre processos (IPC - *Inter-process communication*) e chamada remota de métodos, que permite que um processo possa chamar uma rotina em outro processo, e que é responsável pela identificação do método a ser invocado e da passagem dos argumentos entre os processos [eLinux 2012].

Com relação a memória, o Android introduz um mecanismo (OOM – *Out-of-Memory Handler*) para terminar processos quando na falta de memória, além do ashmem e pmem. O primeiro deles é um alocador de memória compartilhada, com melhor suporte a dispositivos com pouca capacidade de memória. Já o segundo é um alocador de memória de processo, e é utilizado para o gerenciamento de grandes regiões contíguas de memória física compartilhadas entre o espaço do usuários e drivers do kernel [eLinux 2012].

Além das modificações descritas acima, o kernel Linux do Android também recebeu modificações relacionadas ao gerenciamento de energia, devido a própria natureza dos dispositivos móveis, e depuração de erros, com a introdução do *Android Debug Bridge*, um protocolo executado sobre um link USB, além de um sistema de log separado do sistema do Linux e que armazena mensagens no buffer do kernel [eLinux 2012].

4. Dalvik VM

Apesar de as aplicações para Android serem escritas em Java, a Dalvik VM não pode ser considerada uma JVM (*Java Virtual Machine*), pois ela não interpreta Java bytecodes. Ao invés disso, a ferramenta dx transforma os arquivos .class compilados com um compilador Java normal em arquivos .dex, estes específicos para execução na Dalvik VM.

A Dalvik VM é otimizada para dispositivos com pouca memória, e foi desenhada de modo a permitir a execução de várias instâncias ao mesmo tempo de forma eficiente. Ela também deixa a cargo do kernel Linux algumas tarefas, como o gerenciamento de memória e threads e para o isolamento entre processos [DalvikVM 2012].

Uma das medidas tomadas para salvar memória foi a de criar *constant pools* compartilhados para tipos específicos. Uma *constant pool* armazena valores literais constantes utilizados em uma classe, como strings e nomes de classes, interfaces, métodos e atributos. E ao utilizar *constant pools* compartilhados, ao invés de um para cada arquivo .class, como é feito com bytecodes Java, evita-se a duplicação de constantes [Ehringer 2010].

E como aponta [Ehringer 2010], os *constant pools* nos bytecodes Java representam mais de 61% do seu tamanho total, justificando as otimizações que podem ser feitas, tanto para a redução da memória necessária como para reduzir o tempo de inicialização de uma aplicação.

Outra estratégia utilizada para a otimização de aplicativos é o compartilhamento de código entre as instâncias da VM. Isso é feito através de um processo chamado Zygote, que é pai de todas as instâncias da Dalvik VM e é inicializado juntamente com o sistema, carregando todas as classes das bibliotecas que provavelmente serão utilizadas com frequência pelos aplicativos. Assim, quando uma aplicação for iniciada, um comando é enviado ao Zygote, que faz um *fork*, criando um processo que se torna uma nova Dalvik VM, minimizando o tempo de inicialização [Brähler 2010, Ehringer 2010].

Esta estratégia assume que estas classes das bibliotecas compartilhadas serão apenas lidas pelos processos filhos do Zygote. Entretanto, caso um processo resolva escrever em uma dessas classes, a memória compartilhada do Zygote correspondente a classe é copiada para o processo filho para que ele então possam realizar as operações de escrita. Esse comportamento, conhecido como *copy-on-write*, evita a duplicação desnecessária de código sem, entretanto, comprometer a segurança e correto funcionamento das aplicações [Ehringer 2010].

O isolamento entre as aplicações se estende até o *garbage collector* (GC), que é executado em cada VM de forma separada, evitando que GCs de outros processos interfiram na execução de um determinado GC. Cada instância da VM possui a sua própria pilha de memória, além da pilha compartilhada (do Zygote) e a coleta da memória que não é mais utilizada é feita através de *mark bits* que informam ao GC quais objetos devem ser removidos, um mecanismo útil, pois mantém intocada a memória compartilhada do Zygote [Maia et al. 2010]. Antes da versão Gingerbread, toda vez que o GC era executado, as threads do aplicativo eram paradas, inconveniência que foi removida com a introdução do GC concorrente [Dubroy 2011].

5. Framework de Aplicações

Esta camada do *stack* Android é um conjunto de ferramentas que fornecem vários serviços para os aplicativos, visando o reuso de componentes [Android Developers 2012].

Aplicações normalmente são compostas por atividades (*Activity*). No Android uma atividade é uma ação específica que um usuário pode realizar dentro de um aplicativo. Estas ações podem incluir a inicialização de outras atividades, tanto dentro como fora da aplicação a qual ela pertence, através de intenções (*Intent*), criando o que se chama de tarefa. Uma tarefa pode conter várias atividades, que são organizadas em uma pilha na ordem em que foram criadas, permitindo assim que o usuário volte para atividades em que estava anteriormente [Android Developers 2012].

Quando uma uma nova tarefa é iniciada ou o usuário volta para a tela inicial,

a tarefa anteriormente em primeiro plano passa para o segundo plano. Uma tarefa em segundo plano pode voltar ao primeiro plano e várias tarefas podem estar em segundo plano ao mesmo tempo. Entretanto, se muitas tarefas estiverem em segundo plano, o sistema pode começar a destruir atividades para recuperar memória, fazendo com que as atividades percam seus estados, o que não significa que o usuário não possa mais navegar de volta a esta atividade [Android Developers 2012].

Todo o ciclo de vida das atividades, bem como a organização das mesmas em pilhas e tarefas é de responsabilidade do *Activity Manager*, visto na Figura 1. Além dele existem ainda: *Package Manager*, resposável por manter o registro de todas as aplicações instaladas no dispositivo; *Windows Manager*, que gerencia a tela do dispositivo e cria superfícies para as aplicações em execução; *Telephony Manager*, fornece a outras aplicações serviços relacionados com telefonia; *Content Provider*, para o compartilhamento de dados entre aplicações; *Resource Manager*, que provê acesso a tudo aquilo que faz parte de um aplicativo e que não seja código (recursos), como arquivos XML, bitmaps ou outros arquivos, como sons [Khan et al. 2010].

Existe ainda o *View System*, que fornece um conjunto básico de *views* que podem ser utilizadas e extendidas por aplicações, incluindo: listas, caixas de texto, tabelas e botões. E o *Notification Manager*, que permite que uma aplicação notifique o usuário sobre algum evento através de vibração, piscar de LEDs, tocar de algum som ou um ícone na barra de status [Khan et al. 2010].

Além da *Activity* e do *Content Provider*, existem outros dois tipos de componentes de aplicação: os *Services*, que diferentemente de atividades, não possuem uma interface de usuário e são destinados para a execução em segundo plano de operações de longa duração ou para trabalhar com processos remotos, como a comunicação com *web services*; e os *Broadcast Receivers*, que são componentes que respondem a anúncios feitos para todo o sistema, como o anúncio de que a tela se desligou ou de que a bateria está fraca [Android Developers 2012].

6. Conclusão

Ao longo dos anos a computação móvel evoluiu muito, deixando de ser apenas um meio de comunicação através da voz, para se tornar uma mídia que permitiu a criação de uma classe nova de aplicações. Ainda assim, muitas das suas restrições continuam presentes. Este artigo apresentou algumas das estratégias e mecanismos presentes na arquitetura Android e que buscam amenizar estas restrições, principalmente aquelas relacionadas com economia de bateria e memória.

O artigo também fez uma breve introdução aos principais componentes utilizados na construção de aplicações. E muito embora o desenvolvimento de aplicações em Android exija o conhecimento do funcionamento destes componentes, o fato de a linguagem de programação utilizada ser Java e de boa parte das bibliotecas padrão estarem presentes no Android reduzem significativamente a curva de aprendizado necessária.

Referências

Android Developers (2012). The developer's guide. Disponível em: <http://developer.android.com/guide/index.html>. Acesso em: maio 2012.

Brähler, S. (2010). Analysis of the android architecture.

DalvikVM (2012). Dalvik virtual machine. Disponível em: <http://www.dalvikvm.com/>. Acesso em: maio 2012.

Dubroy, P. (2011). Memory management for android apps. Google I/O 2011. Disponível em: <http://bit.ly/Md5QDX>. Acesso em: maio 2012.

Ehringer, D. (2010). The dalvik virtual machine architecture. Disponível em: <http://www.kiddai.com/NCTU/ebl/dex/The_Dalvik_Virtual_Machine.pdf>. Acesso em: maio 2012.

eLinux (2012). Android portal. Disponível em: <http://elinux.org/Android_Portal>. Acesso em: maio 2012.

Gartner (2012). Gartner says worldwide sales of mobile phones declined 2 percent in first quarter of 2012; previous year-over-year decline occurred in second quarter of 2009. Disponível em: <http://www.gartner.com/it/page.jsp?id=2017015>. Acesso em: maio 2012.

Google (2012). Introducing google play: All your entertainment, anywhere you go. Disponível em: <http://googleblog.blogspot.com.br/2012/03/introducing-google-play-all-your.html>. Acesso em: maio 2012.

Khan, S., Khalid, S. H., Nauman, M., Khan, S., and Alam, M. (2010). Analysis report on android application framework and existing security architecture. Technical report, Security Engineering Research Group, Pakistan.

Lunden, I. (2012). Google play about to pass 15 billion app downloads? pssht! it did that weeks ago. TechCrunch. Disponível em: <http://techcrunch.com/2012/05/07/google-play-about-to-pass-15-billion-downloads-pssht-it-did-that-weeks-ago/>. Acesso em: maio 2012.

Maia, C., Nogueira, L. M., and Pinho, L. M. (2010). Evaluating android os for embedded real-time systems. Technical report, Polytechnic Institute of Porto (ISEP-IPP), Porto, Portugal.

McDermott, P. (2008). Porting android to a new device. Disponível em: <http://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/Porting-Android-to-a-new-device/>. Acesso em: maio 2012.