Seminário de S.O - Google Android

Diogo de Campos
João Paulo Pizani Flor
Maurício Oliveira Haensch
Pedro Covolan Bachiega

19 de novembro de 2008

1 Introdução

O presente trabalho foi apresentado como seminário na disciplina de Sistemas Operacionais I, durante o segundo semestre de 2008. Nosso tema escolhido foi a plataforma Android, desenvolvida pela Google focando dispositivos móveis. A plataforma Android inclui tanto sistema operacional customizado, como middleware e aplicativos-chave para estes dispositivos.

O Android usa o kernel 2.6 do *Linux* como base, com várias modificações para se adaptar melhor aos dispositivos-alvo. Essas modificações incluem: sistema de comunicação inter-processos, otimizações no sistema de gerenciamento de energia e no gerenciamento de memória.

O foco principal do seminário foi expor as adaptações mencionadas, e também alguns outros componentes da plataforma Android em nível de usuário, como por exemplo a biblioteca C padrão customizada (*Bionic* libc), a máquina virtual Dalvík (que roda um formato especial de bytecodes), entre outros.

Inicialmente, fazemos uma breve introdução à plataforma Android: como surgiu, quem foram os envolvidos e a arquitetura básica do sistema. Então passamos a detalhar cada um dos itens pesquisados; tanto as modificações realizadas no núcleo do sistema operacional, quanto às bibliotecas e frameworks de aplicação.

2 A Plataforma Android

2.1 Histórico

Em 2005, surgiram rumores de a Google estaria disposta a entrar no mercado de dispositivos móveis, quando ela comprou a Android Inc., uma pequena companhia do Vale do Silício que fabricava software para telefones celulares. Ainda era incerto, porém, qual era a função que a Google iria ter nesse mercado. Em setembro de 2007, estudos descobriram que a Google tinha submetido vários pedidos de patentes.

Em Novembro do mesmo ano Eric Schmidt, CEO da Google, veio a público para desmentir os rumores de que a empresa iria lançar qualquer tipo de aparelho móvel. Em seu discurso Schmidt afirmou que o esforço da empresa era mais ambicioso do que qualquer "Google Phone". O produto desenvolvido era na verdade uma plataforma de software que iria estar presente em diversos modelos de aparelhos. No mesmo mês a Open Handset Alliance lançou o Android.

2.2 O que é o Android

Android é uma completa pilha de componentes de software, desenvolvida para dispositivos móveis, que inclui sistema operacional, bibliotecas e frameworks de middleware e aplicações-chave [5]. A plataforma Android é e será mantida por um grupo de mais de 30 empresas, a Open Handset Alliance. São empresas fabricantes de dispositivos móveis (celulares, PDA's, Internet tablets), de semicondutores e de software, que se aliaram na criação da primeira plataforma móvel completa, aberta e livre.

2.3 Características básicas

A parte de baixo nível da plataforma Android e que é dependente de hardware usa como base kernel do Linux 2.6. Ele é reponsável pelas tarefas fundamentais de sistema, como segurança, gerenciamento de memória, gerenciamento de processos, pilha de protocolos de rede e modelo de drivers. Ele também age como uma camada de abstração entre o hardware e os outros componentes de plataforma, que rodam em espaço de usuário.

Várias alterações foram realizadas no Linux para que o sistema ficasse melhor adaptado às características dos dispositivos móveis, e essas modificações serão detalhadas mais à frente. Elas incluem device drivers novos, adições ao sistema de gereciamento de energia (*wake locks*) e um sistema que permite finalizar processos de maneira criteriosa quando há pouca memória disponível (*lowmem killer*).

Já os componentes independentes de hardware constitusubsectionem a maioria da plataforma Android, e incluem uma biblioteca padrão C customizada, codecs para inúmeros formatos multimídia, um engine de browser (*Webkit*) também usado pelo Safari, ambiente gráfico e gerenciador de pacotes.

O coração da plataforma Android é certamente a máquina virtual Dalvík, que roda software escrito na linguagem Java e compilado num formato especial de bytecodes, o .dex (Dalvík Executable). A Dalvík é uma máquina de registradores, em oposição às tradicionais máquinas Java, que usam o modelo de máquina de pilha. A Dalvík é otimizada para sistemas com pouca memória e

não usa *Just-in-time compilation*. Não se pode dizer precisamente que a Dalvík é uma máquina virtual Java, pois ela não suporta arquivos .class. Existe, porém, uma ferramenta (*dx*), que faz a ligação e converte vários arquivos .class para um único .dex.

3 Arquitetura

A figura 1, obtida de [5] ilustra quais são e como se relacionam as camadas de componentes de software que constituem o Android. Essas camadas abrangem desde o núcleo do sistema operacional Linux, passando por middleware e ambiente de execução virtualizado, até as aplicações em si.

Nessa seção detalharemos cada uma dessas camadas, expondo suas responsabilidades e a relação que têm com as demais.

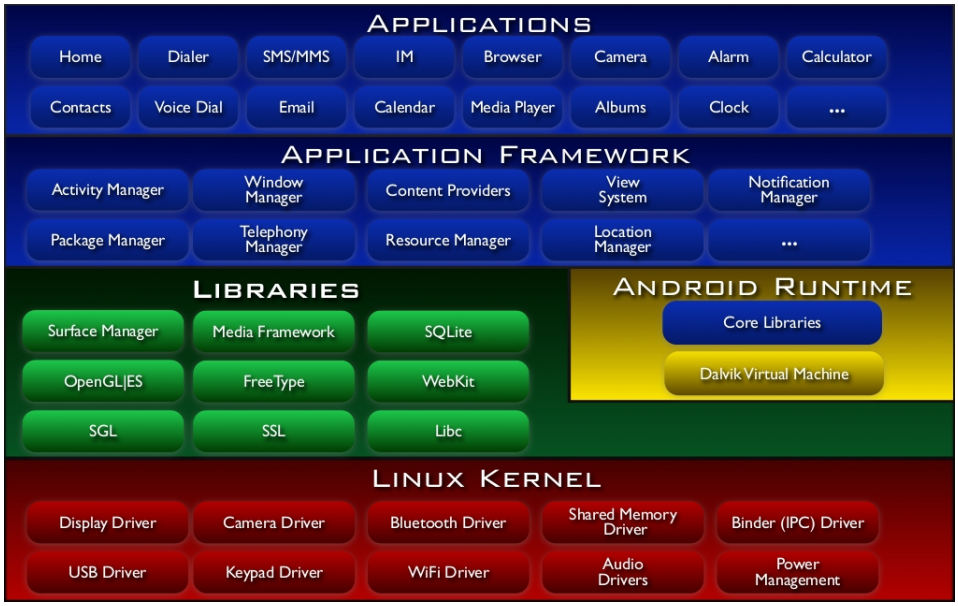

Figura 1: Arquitetura lógica em camadas da plataforma Android

3.1 Linux 2.6

O Android é baseado no Kernel 2.6 do Linux, mas não é Linux! Várias funções do Kernel são utilizadas diretamente pelo Android, mas várias modificações foram feitas para otimizar a memória e tempo de processamento das aplicações [4]. Linux foi escolhido por já ter uma boa quantidade de device drivers sólidos, além de bons sistemas de gerenciamento de memória e processos, que são utilizados pelo Android. Além destas funções, o Kernel também é usado para cuidar de alguns serviços de segurança e rede, e serve como uma camada de abstração entre o software, e o hardware. O desenvolvedor de aplicações não irá programar nesta camada do Android, e só fará uso indireto do Kernel através de APIs de níveis superiores.

3.2 Framework de aplicação

O framework para aplicações Android [1] tem as mesmas APIs que as aplicações chave do Android. Este framework foi criado para facilitar a vida dos programadores, simplificar o reuso de componentes e abstrair boa parte dos proced-

imentos necessários para se fazer uma aplicação funcionar. Uma grande vantagem do framework de aplicação do Android é que cada aplicação pode publicar suas capacidades para serem usadas por outras aplicações que estiverem rodando.

Entre estas APIs, temos as seguintes:

Location Manager Usado em aplicações que precisam saber a posição geográfica do usuário, como por exemplo, GPS e aplicações para verificar o clima e tempo da região.

Telephony Manager Camada de abstração para lidar com os serviços de telefonia do celular. Informações sobre a capacidades e restrições do dispositivo podem ser obtidas através desta API. Afinal, um celular *ainda* serve para fazer ligações.

Window Manager Gerente simples que permite criar e obter informações sobre a janela de exibição da aplicação.

Content Providers Os Content Providers permitem que uma aplicação torne seus dados públicos, e também que ela acesse dados de outras aplicações. Content Providers são usados em aplicações que, por exemplo, precisam ler o banco de dados de contatos do usuário, ou tabelas criadas por outras aplicações. Quase todo tipo de dado é compartilhável, como áudio, vídeo, imagens e texto.

Resource Manager Todos os recursos de uma aplicação são separados de seu código, como imagens, strings e arquivos XML. Esses recursos são otimizados para ocupar menos espaço e demorar menos tempo para carregar. Para facilitar o acesso do desenvolvedor a seus recursos, foi criado o Resource Manager, que permite obter e modificar seus dados externos facilmente.

Notification Manager API que permite que as aplicações exibam notificações na tela do dispositivo, ou ativar LEDs, luzes ,sons ou vibração.

Activity Manager Cada atividade no systema é gerenciada através de uma pilha de atividades, quando uma nova é criada, esta vai para o topo da pilha e se torna a *running activity*. As atividades tem 4 estados básicos:

Ativa: se ela está na camada principal da tela.

Pausada: se ainda é visível mas não está selecionada (*background*). Uma atividade pausada continua viva, mas em casos extremos pode ser finalizada pelo sistema.

Parada: se não é mais visível (esta coberta por outra aplicação). Suas informações e estados são mantidos, mas são frequentemente terminadas em caso de necessidade de memória.

Inativa: quando uma atividade estiver pausada ou parada, esta pode ser terminada pelo sistema, e caso o usuário a queira ativar novamente, ela deve ser carregada e seu estado anterior deve ser restaurado.

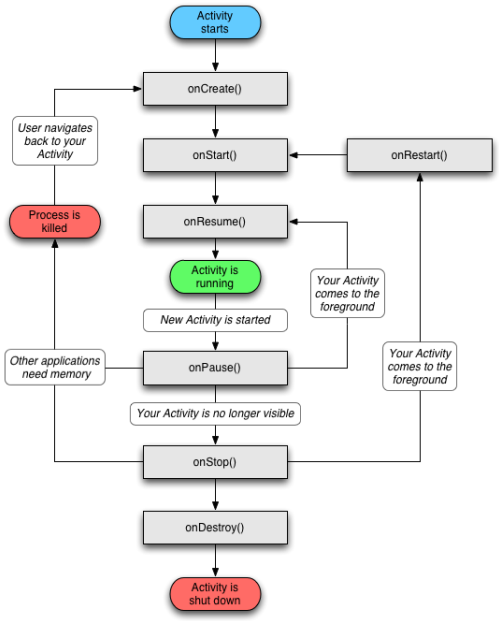

Figura 2: O ciclo de vida de uma activity

3.3 Máquina Virtual Dalvík

Durante o *boot* do sistema operacional do Android, é criado uma instância especial da Dalvík (*zygote*) que será responsável pela criação de várias instâncias da máquina virtual, uma para cada aplicação executada. Esta máquina roda arquivos .dex, um formato de bytecode desenhado para ocupar menos espaço e carregar rapidamente. O Android permite o uso de *multi-threads* que é suportado pela máquina virtual, junto com outras utilidades, como o Garbage Collector, também otimizado para rodar em aplicativos com poder de processamento e memória reduzidos.

3.4 Aplicações

A camada mais alta da arquitetura da plataforma Android contém uma śerie de aplicações chave, incluindo cliente de e-mail, SMS, mapas, calendário, browser, agenda de contatos, entre outros. Todas as aplicações são escritas em Java (por enquanto, há esforços para possibilitar a escrita de aplicativos em Python e em Ruby).

As aplicações podem facilmente embutir páginas Web, rodam simultaneamente e o usuário pode alternar facilmente entre as aplicações ativas. Uma outra característica interessante é o fato das aplicações não rodarem diretamente no Android, mas sim numa instância da máquina virtual Dalvik. O código Java é convertido para um arquivo .dex, que nada mais é que um bytecode dessa máquina virtual. Isso faz com que o Android possa rodar qualquer linguagem, desde que convertida para o bytecode da Dalvik.

4 As modificações no Linux 2.6

4.1 O sistema de IPC Binder

As aplicações e serviços que rodam no Android podem rodar em processos separados, mas muitas vezes precisam comunicar-se e compartilhar dados. Nos sistemas operacionais convencionais existem vários mecanismos disponíveis para realizar *Comunicação inter-processos* (IPC). Nos sistemas Unix-like, por exemplo, podem ser usados pipes (nomeados ou anônimos), ou mesmo infraestruturas mais sofisticadas, como o D-Bus, do projeto Freedesktop.org e muito utilizado nas distribuições Linux atuais.

Nos sistemas orientados a objetos a grande questão a ser enfrentada na comunicação inter-processos é a *serialização* dos objetos usados na comunicação. Isso é a causa do maior overhead de processamento. Além disso, a necessidade de comunicação inter-processos também aumenta a chance de ocorrerem falhas de segurança.

A figura 3 mostra um exemplo de uma aplicação utilizando a API do Binder, que está na camada de Framework de aplicação, para realizar comunicação com outra aplicação. O Binder é responsável por obter e expor à aplicação requerente a interface de serviços da outra. Ele também realiza a tarefa de serialização dos objetos enviados e recebidos nas chamadas.

4.2 Android Interface Description Languagem

AIDL (Android Interface Description Language) é uma linguagem de descrição de interfaces para aplicações criada para facilitar a comunicação entre aplicações no Android. Um dos da criação da AIDL é que ela possibilita aplicativos escritos em diferentes linguagens de programação se comunicarem de uma maneira uniforme [3].

O Binder permite que um processo acesse a API de outros, mas para isso é preciso que estas estejam publicadas no Binder. Como este trabalho é relativamente complicado, para ajudar os desenvolvedores, foi criada a AIDL, que é fácil de usar. Agora, basta o programador descrever sua interface externa em AIDL e colocar este arquivo junto com a aplicação, ele então será processado e inserido no Binder para que as outras aplicações tenham acesso as APIs escolhidas.

4.3 A biblioteca padrão C Bionic

Uma nova biblioteca C padrão foi desenvolvida especialmente para o Android. Esta biblioteca chama-se *Bionic*. O desenho de uma *nova libc* foi necessário por três principais motivos [2]:

Licença A Google queria manter a GPL longe do espaço de usuário (Userspace). A Bionic usa licença BSD.

Tamanho A biblioteca completa deveria ser carregada com cada processo, por isso precisava ser pequena. A Bionic tem cerca de 200K, aproximadamente metade de tamanho da *glibc*.

Velocidade O poder de processamento limitado dos dispositivos móveis obriga a biblioteca a ser rápida, sob pena de se tornar absolutamente inútil. A

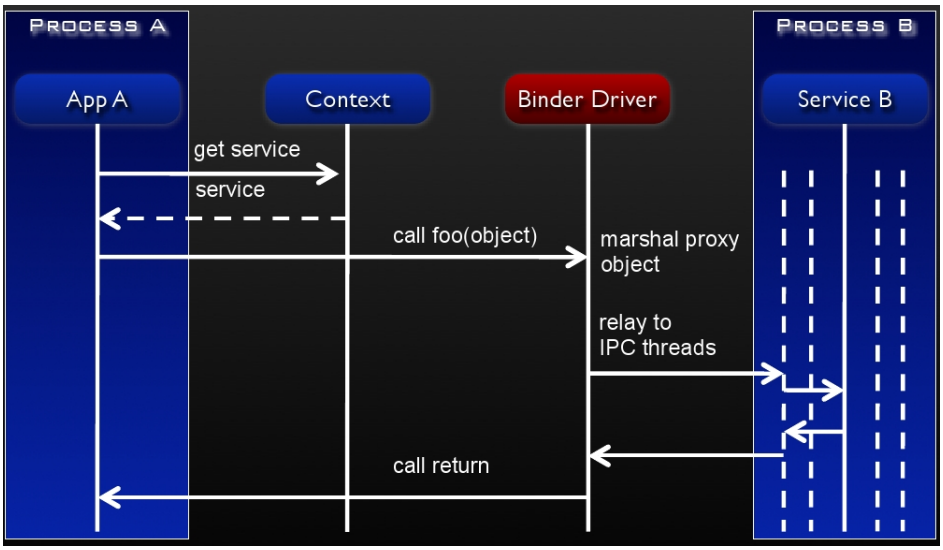

Figura 3: Uma aplicação realizando comunicação inter-processos com o Binder

> Bionic tem implementações geralmente pequenas e otimizadas dos algoritmos de biblioteca padrão, incluindo uma implementação rápida e pequena do pacote *pthread*.

A Bionic suporta nativamente vários serviços específicos à plataforma Android, como propriedades do sistema e logging. Ela *não possui* todas as funcionalidades previstas nos padrões POSIX. Alguns exemplos de funcionalidades não-incluídas são exceções C++ e wide chars.

Fica claro, portanto, que a Bionic libc não é compatível com a glibc (implementação GNU da libc). Todo o código nativo precisa ser compilado e ligado contra a bionic, e não a glibc.

4.4 O sistema de gerenciamento de energia

A plataforma Android provê um sistema de gerenciamento de energia próprio, que roda sobre o sistema de gerenciamento de energia do Linux 2.6. O gerenciamento de energia é requisitado pelos aplicativos utilizando primitivas de programação chamadas *Wake locks*, que estão presentes na API Java do Android.

As Wake locks são um mecanismo semelhantes às travas que os S.O convencionais utilizam para prover acesso a recursos como arquivos. Um aplicativo deve requisitar um Wake lock para demonstrar ao Power Manager que precisa de um certo recurso ligado durante um trecho da execução.

O driver de gerenciamento de energia do kernel irá, então, checar periodicamente por todos os dispositivos que não estiverem travados por nenhuma aplicação, e irá desligá-los. Devido à essa política agressiva, os Wake locks devem ser utilizados com cautela pois o seu mau uso pode ocasionar desperdício de energia, ao contrário da economia desejada.

Na figura 4 damos um exemplo do procedimento através do qual uma aplicação pode requisitar um Wake lock. A aplicação utiliza a API do PowerManager para

fazer a requisição. Este, após verificar a validade da requisição, registra o pedido junto ao driver de gerenciamento de energia do kernel. O kernel periodicamente checa pelos *constraints* correntes de energia, e desliga os recursos que não estão sendo utilizados por nenhuma aplicação.

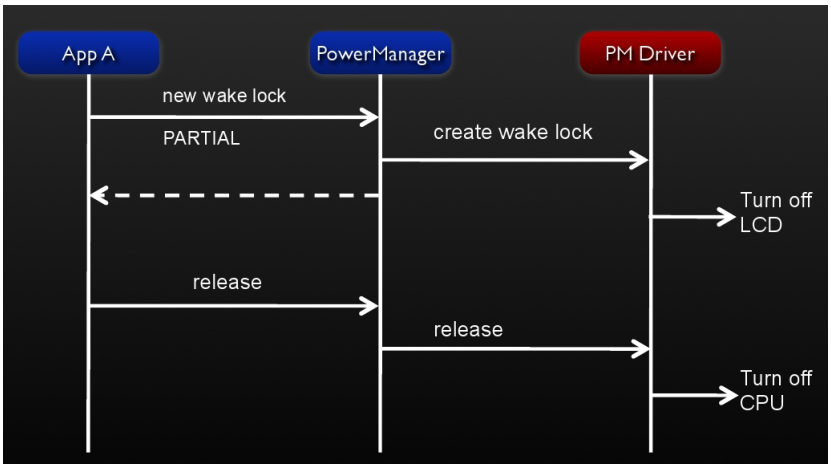

Figura 4: Um exemplo de um aplicação requisitando e liberando um wake lock

5 A máquina virtual Dalvík

As aplicações da plataforma Android rodam em instâncias da máquina virtual Dalvik. A Dalvik é uma máquina de registradores, projetada e escrita por Dan Bornstein com outros engenheiros da Google especialmente para o Android. Ela foi projetada para rodar em sistemas com baixa frequência de CPU, pouca memória RAM disponível e SO sem espaço de swap. Outra característica dessa máquina virtual é capacidade de serem rodadas diversas instâncias ao mesmo tempo, deixando a cargo do Sistema Operacional o gerenciamento de memória, isolamento de processos e suporte a threads.

A Dalvik não é uma máquina virtual Java, pois ela utiliza seu próprio bytecode, no formato .dex, ao invés do bytecode próprio do Java. O Android Software Development Kit contém uma ferramenta, chamada dx, que transforma arquivos .class de Java para o formato de bytecode da Dalvik. Isso permite que a plataforma venha a suportar outras linguagens assim que surgirem ferramentas para conversão para o formato .dex.

Na inicialização do sistema, é criado um processo para a máquina virtual denominado Zygote. A partir desse processo, outras máquinas são instanciadas com fork() quando necessárias para rodas outras aplicações. No Zygote, ficam as bibliotecas compartilhadas em modo somente leitura, e todos os processos de aplicações tem partes de sua memória mapeadas nesta região do Zygote. Os processos do Android têm, além de suas pilhas, coletores de lixo separados. Eles devem ser independentes porém devem também respeitar o compartilhamento. A figura 5 demonstra o layout do processo zygote e das instâncias adicionais da Dalvík.

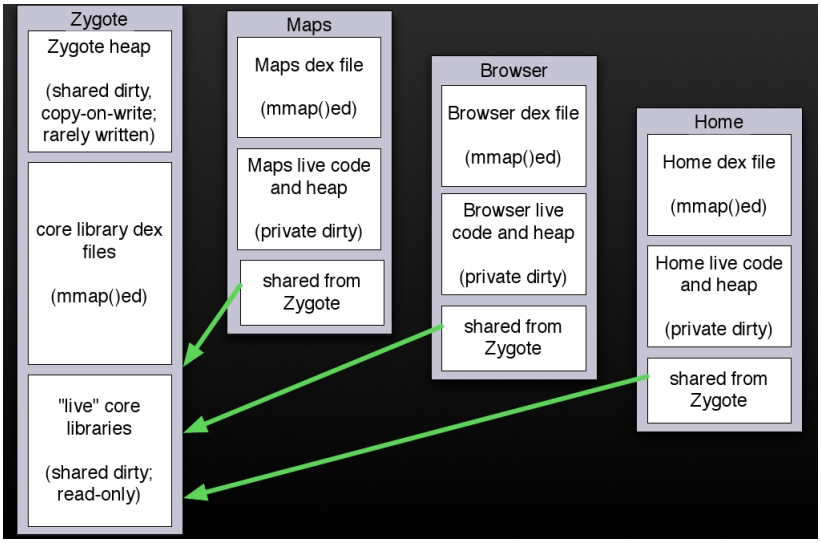

Figura 5: Layout do processo zygote e das outras instâncias da Dalvík

Referências

[1] Android application framework. http://code.google.com/android/reference/android/. Página oficial do projeto Android.

[2] The android bionic libc. http://discuz-android.blogspot.com/2008/10/google-android-native-libc-bionic.html.

[3] The android interface description language. http://code.google.com/android/reference/aidl.htmll. The Android Reference Information.

[4] How android works. http://blogs.zdnet.com/Burnette/?p=515.

[5] What is android? http://code.google.com/android/what-is-android.html. Página oficial do projeto Android.