UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DAVI AUGUSTO NEVES LEITE
GIOVANI CANDIDO

VISÃO GERAL DA ARQUITETURA DO SISTEMA ANDROID: DESVENDANDO A PLATAFORMA MÓVEL DA GOOGLE

BAURU
2021

DAVI AUGUSTO NEVES LEITE
GIOVANI CANDIDO

VISÃO GERAL DA ARQUITETURA DO SISTEMA ANDROID: DESVENDANDO A PLATAFORMA MÓVEL DA GOOGLE

Monografia apresentada ao Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru, como requisito parcial para aprovação na disciplina de Sistemas Operacionais I.
Orientador: Prof. Dr. Antonio Carlos Sementille

BAURU
2021

Dedicamos este trabalho a todas as pessoas que ajudaram e continuam ajudando a iluminar nossas vidas. Em especial, dedicamos aos nossos amigos, familiares e parceiras.

Agradecimentos

Agradecemos a Deus por toda a compaixão e por todo o apoio que tivemos ao longo de nossas vidas, independentemente de nossas faltas.

Aos nossos familiares que nos concederam um suporte incondicional assim como o amor mais pleno e genuíno que possa existir.

Aos nossos amigos por todo o conforto que nos ofereceram e por cada bom momento que pudemos partilhar.

As nossas namoradas que sempre nos encheram de alegria, nos puxando para cima quando achávamos estar bem baixo.

A todos os professores que estiveram presentes em nossa jornada e que se empenharam para que pudéssemos ser o que somos hoje.

Agradecemos especialmente a Laiz Fernanda Alves Ferreira, Luis Henrique Morelli e Luiz Fernando Merli Sementille, pela presença inesquecível em nossas vidas.

Agradecemos a todas as pessoas que contribuíram direta ou indiretamente para o nosso desenvolvimento pessoal e para a nossa formação acadêmica.

Forças além do seu controle podem tirar tudo o que você tem, exceto uma coisa: sua liberdade de escolher como irá responder a qualquer situação. Você não pode controlar o que acontece em sua vida, mas você sempre pode decidir como irá se sentir e o que irá fazer a respeito.

Viktor Emil Frankl

Resumo

O Android é um sistema operacional multitarefa voltado para dispositivos móveis, como celulares e tablets. O objetivo central do trabalho consiste em definir, tanto no geral quanto no específico, as camadas da arquitetura do Android, mostrando a interrelação de funcionamento das mesmas. Primeiramente, procurou-se artigos relacionados ao assunto nas mais famosas bases de dados a respeito de computação, como IEEE e ACM. Contudo, não se obteve resultados satisfatórios, necessitando utilizar a ferramenta de busca do Google Acadêmico e, dessa forma, pode-se encontrar artigos e livros que tratavam a respeito desse assunto, como o livro "Sistemas Operacionais Modernos" dos autores renomados Andrew Tanenbaum e Herbert Bos. Além disso, outra fonte fundamental para a coleta de dados foi a documentação oficial do sistema. Diante disso, foi possibilitado extrair o máximo das informações a respeito das camadas da arquitetura Android. Por fim, as informações trazidas nesta monografia possibilitam o entendimento, de maneira fácil, a respeito do funcionamento interdependente e relacional das camadas e dos componentes de cada uma da arquitetura Android.

Palavras-chave: Android; Arquitetura; Sistema Operacional.

Lista de figuras

Lista de quadros

Sumário

1 Introdução

O Android é um sistema operacional multitarefa voltado para dispositivo móveis, como tradicionalmente celulares e tablets, sendo adaptado para ambientes com recursos mais modestos do que os encontrados em computadores ou microcomputadores (MCHOES; FLYNN, 2013). Segundo Brahler (2010), o sistema em questão suporta serviços usuais de telefonia e diversos outros serviços de computação, além de possibilitar a execução de aplicações distribuídas entre múltiplos fluxos de controle ou entre múltiplos processos, como em sistemas desktop.

1.1 Contexto e Problema

De início, Bootlin (2019) conta que, em 2003, Andy Rubin fundou a Android Inc., empresa que atuava no ramo de software para celulares inteligentes, e tinha como plano, mantido em segredo, criar um sistema operacional que, segundo Tanenbaum e Bos (2016), fosse capaz de fazer os dispositivos móveis mais espertos. Conforme Gunasekera (2012) elucida, a empresa despertou o interesse da Google que, em 2007, realizou a aquisição da Android Inc., uma vez que a gigante tecnológica desejava, de acordo com Krajci e Cummings (2013), integrar seus serviços ao mundo móvel e lucrar por meio do alcance mais abrangente de anúncios.

Por consequência, segundo Krajci e Cummings (2013), o sistema operacional está presente em milhões de dispositivos móveis ao redor do mundo e o sucesso pode ser justificado tanto pelo conjunto de empresas envolvidas no projeto quanto pela sua natureza livre. Ainda de acordo com Krajci e Cummings (2013), a Google criou uma parceria com Intel, Qualcomm, Motorola, Samsung e outras gigantes tecnológicas para promover o desenvolvimento e a manutenção do sistema. Ademais, Tanenbaum e Bos (2016) apontam que o código-fonte do sistema é mantido em um repositório público da Google, possibilitando que qualquer desenvolvedor possa contribuir com o projeto, sem contar que cada fabricante de dispositivos ganha a liberdade de alterar o sistema e criar uma versão com os seus próprios produtos.

Ainda, segundo Page (2013, tradução nossa), o Android tem "[...] uma parceria global com mais de 60 fabricantes [de dispositivos móveis]; mais de 750 milhões de dispositivos ativos globalmente; e 25 bilhões de aplicativos já foram baixados da Google Play". Gunasekera (2012) afirma que, pelo primeiro semestre de 2011, o Android já representava metade do mercado de sistemas operacionais para celulares, o que ressaltava o potencial do sistema, tendo em vista que o Android havia sido lançado a menos de três anos atrás.

Por outro lado, antes do surgimento do Android, conforme Page (2013) explica, cada celular lançado no mercado contava com um sistema operacional especialmente projetado para aquele dispositivo e, por consequência, os desenvolvedores eram obrigados a criar aplicações

exclusivas para cada celular. Como bem lembra Krajci e Cummings (2013), a plataforma Android não foi projetada com algum dispositivo específico em mente, mas sim com o propósito de ser um ambiente comum a muitos dispositivos, proporcionando uma padronização benéfica aos fabricantes e aos desenvolvedores, uma vez que não haveria nem a preocupação de criar sistemas novos para cada novo aparelho, nem a preocupação de criar aplicativos exclusivos.

De acordo com Tanenbaum e Bos (2016), dentre as metas iniciais do projeto da antiga Android Inc., o desejo de criar um sistema que pudesse ser amplamente utilizado já estava presente e a opção por utilizar o núcleo Linux no Android foi determinante para que o desejo fosse realizado com o tempo. Outrossim, os autores salientam que, uma das vantagens do uso do Linux, foi a capacidade de aproveitar recursos já existentes no núcleo em questão, mesmo que seja de uma maneira pouco usual ou com algumas modificações e acréscimos.

Além do kernel já conhecido pela comunidade de desenvolvedores, outro fator que destaca o Android é a utilização da linguagem Java no alto nível. Ainda segundo os autores, o sistema Android tem alguns mecanismos próprios em código nativo, escritos em C/C++ que são as linguagens presentes no núcleo, porém a maior parte dos componentes do sistema, a interface de programação de aplicações (API, em inglês) e as aplicações utilizam a popular linguagem Java. Vale notar que a Java é muito utilizada por ser, entre outras coisas, retroativa e orientada à objetos.

Tendo em vista todo o exposto, é visível que o projeto Android necessita de uma documentação bem detalhada para possibilitar o entendimento de toda a arquitetura e funcionamento do sistema, uma vez que o sistema é composto de muitas partes interdependentes. Além de que, o desenvolvimento da plataforma ocorreu, e tem ocorrido, em um ritmo bem acelerado. De acordo com Android Developers (2021), o Android já está em sua trigésima versão, lançada em setembro de 2020, sendo que a primeira versão, conforme Krajci e Cummings (2013) afirmam, foi lançada em setembro de 2008; considerando o intervalo de 22 anos entre o lançamento da primeira e o da última versão, foi lançada, no mínimo, uma versão por ano.

Por conseguinte, como bem lembra Brahler (2010), a alta mutabilidade da arquitetura, seja por otimizações ou pelo emprego de novas técnicas e tecnologias, gera uma obsolescência muito rápida das informações, de forma que poucas fontes consigam acompanhar as novidades e manter um conteúdo confiável e atualizado em relação ao funcionamento da plataforma Android. Entretanto, muito embora exista essa adversidade, existem alguns livros e artigos que trazem conceitos empregados desde versões mais antigas até as mais atuais, além de existir, segundo Brahler (2010), uma extensiva documentação oficial sempre, bem atualizada, encontrada nos sites Android Developers e Android Open Source Project.

O presente trabalho consiste em um estudo bibliográfico a respeito da arquitetura do sistema Android, tencionado trazer informações confiáveis de livros, artigos e da documentação oficial da Google. Ademais, o estudo propõe descomplicar o material encontrado, que geralmente tem um teor extremamente técnico, por meio da análise somente dos principais componentes da

estrutura do sistema, abstraindo alguns pormenores que não sejam essenciais ao entendimento.

Para mais, o trabalho está estruturado em quatro capítulos. O primeiro capítulo traz uma breve explicação sobre o sistema Android, comentando um pouco da história, do mercado e de alguns componentes da arquitetura, além dos objetivos e da justificativa do trabalho. O segundo capítulo é subdivido em três partes: História do Android, que aborda desde o surgimento e criação do sistema até os dias atuais; Histórico de Versões, que passa por cada versão do sistema operacional e evidencia as mudanças mais marcantes; e Arquitetura do Android, que é o enfoque do trabalho e abarca os principais componentes do sistema. O terceiro capítulo demonstra as etapas de realização do trabalho, como a coleta e análise dos dados. E, por fim, o quarto capítulo retoma a discussão inicial e averigua se os objetivos foram cumpridos.

1.2 Justificativa

O presente estudo se justifica pela obsolescência de outros materiais acerca do tema, já que, devido à rápida evolução do Android, há uma dificuldade em filtrar livros e artigos que foram capazes de acompanhar as mudanças que têm ocorrido na estrutura do sistema. Ademais, é possível afirmar que não há somente a questão da dificuldade em encontrar conteúdo atualizado, mas há também uma escassez de livros e artigos científicos que abordem toda a arquitetura do sistema. Ao pesquisar por livros sobre a arquitetura da plataforma, a grande maioria dos resultados são livros práticos sobre a programação de aplicações para o sistema. Ademais, em termos de artigos científicos, pesquisas sobre a arquitetura nas bases de dados IEEE, ACM, SciELO e CAPES, não resultam em artigos que de fato tenham como proposta realizar um estudo ou análise da arquitetura do Android. Na seção da metodologia, a coleta de dados empregada no trabalho em questão será melhor discutida.

1.3 Objetivo Geral

Esmiuçar a função de cada camada da arquitetura do sistema Android, desde o núcleo até o mais alto nível, visando evidenciar como o trabalho conjunto dos principais componentes de cada camada colabora para a experiência do usuário final.

1.4 Objetivos Específicos

Expor dados a respeito do surgimento e da evolução do sistema até o momento vigente, tendo em vista fornecer um panorama histórico da plataforma.

Filtrar quais informações são de fato essenciais para a compreensão do funcionamento do sistema Android, além de filtrar quais conceitos são atuais e ainda aplicados nas versões mais recentes da plataforma.

Discorrer sobre a arquitetura do sistema, tencionado examinar suas principais camadas e os principais integrantes de cada camada.

Explicar as responsabilidades de cada componente, evidenciando as relações de interdependência entre os vários componentes do sistema.

2 Revisão de Literatura

O Android, conforme explicam Mchoes e Flynn (2013), é um sistema operacional que está presente em milhões de dispositivos móveis ao redor do mundo. Ainda segundo os autores, o sistema, que utiliza o kernel Linux, é de código aberto e tem partes do código-fonte disponíveis na internet (MCHOES; FLYNN, 2013). Embora o sistema seja comumente empregado em celulares e tablets, a tecnologia não foi projetada para algum fabricante específico. Assim, Krajci e Cummings (2013) afirmam que o Android possibilitou um ambiente comum a muitos dispositivos, fazendo com que os desenvolvedores pudessem criar aplicações amplamente utilizadas e focar mais na experiência dos usuários.

2.1 História do Android

De acordo com Krajci e Cummings (2013), a história do sistema tem início com a criação da empresa Android Inc. por Andy Rubin e alguns de seus colegas de trabalho em 2003. Bootlin (2019) e Tanenbaum e Bos (2016) contam que a empresa, inicialmente, planejava atuar no ramo de câmeras digitais, mas, por conta do mercado maior, decidiu partir para a telefonia móvel, visando criar um sistema operacional que pudesse fazer os dispositivos móveis mais espertos. O sistema em questão deveria ser capaz de gerenciar o celular "apesar das limitações de energia da bateria, do tamanho da CPU e do reduzido espaço de memória" (MCHOES; FLYNN, 2013, p. 498, tradução nossa).

A chave para o desenvolvimento do sistema Android, segundo Tanenbaum e Bos (2016), foi a compra da Android Inc. pela Google, trazendo não só o suporte financeiro necessário, mas também uma série de serviços em nuvem que já existiam na época, como o Gmail e o YouTube. Além disso, a Google também foi a responsável por encomendar o famoso logo da plataforma, um inseto robótico na cor verde, que tinha o propósito de promover o sistema entre desenvolvedores e, segundo Blok ([20–]), visava criar um impacto semelhante ao causado pela mascote do Linux, o Tux.

Ademais, Tanenbaum e Bos (2016) afirmam que a maior parte do sistema, de fato, foi construída quando o pequeno time da Android Inc. já estava operando junto com a Google, e Mchoes e Flynn (2013) contam que a sacada dos engenheiros foi utilizar o Linux como o núcleo do sistema operacional, uma vez que boa parte dos mecanismos como os de gerenciamento de processos, de memória, de armazenamento etc. poderiam ser reaproveitados, se necessário, com algumas modificações. Embora C e C++ constituam boa parte da plataforma, Tanenbaum e Bos (2016) reforçam que, em termos de linguagem alto nível, a interface gráfica e a porção referente às APIs do sistema foram desenvolvidas em Java.

Por conseguinte, foi criado o Android Open Source Project (AOSP), o repositório de código aberto responsável pelo desenvolvimento e pela manutenção do Android. De acordo com Tanenbaum e Bos (2016), o repositório, liderado pela Google, tem como propósito disponibilizar a documentação e o código-fonte de qualquer versão do sistema operacional que venha a existir, permitindo que qualquer desenvolvedor possa criar uma versão personalizada com softwares nativos de terceiros.

No entanto, ainda que a Google tenha sido crucial, a multibilionária não foi a única empresa empenhada no desenvolvimento da plataforma. Conforme Krajci e Cummings (2013) apontam, em 2007, a Google com algumas fabricantes de celulares e outras empresas tecnológicas formaram a Open Hanset Alliance (OHA), uma aliança que visava fomentar o mercado dos aparelhos móveis e que focava na criação do sistema Android. Outrossim, segundo Bootlin (2019), a aliança, ainda ativa, conta com empresas como: Intel, Qualcomm, Nvidia, Motorola, HTC, Sony, Samsung e T-Mobile; todas interessadas no padrão aberto e universal gerado pelo Android.

Tanenbaum e Bos (2016) expõem que, no início do projeto em 2006, o sistema era desenvolvido com o auxílio de um simulador; entretanto, com o passar do tempo, um celular inteligente intitulado Sooner, que contava com teclado ao invés do toque na tela, foi utilizado para os testar o sistema. Até que, segundo Callaham (2020), a Google adotou como alvo o dispositivo HTC Dream e, em 2007, lançou a primeira versão beta do kit de desenvolvimento de software (SDK) do Android.

Para mais, em conformidade com Tanenbaum e Bos (2016), esta versão beta do SDK era composta do seguinte conjunto de ferramentas: um emulador de dispositivo de hardware capaz de executar o Android, atualmente denominado Android Emulator; toda a documentação da API do sistema; e um ambiente de desenvolvimento integrado, atualmente denominado Android Studio.

Em 2008, houve o lançamento comercial do smartphone HTC Dream, também denominado T-Mobile G1, munido de toque na tela, de um teclado físico de deslizar, de 3G, de um acelerômetro, de um GPS, de um compasso e do sistema operacional Android 1.0 (CALLAHAM, 2020; TANENBAUM, BOS, 2016). O Android 1.0, segundo Krajci e Cummings (2013) e Callaham (2020), trouxe serviços essenciais do SO, além de várias aplicações que são utilizadas até os dias atuais, como Gmail, Google Maps (com StreetView), Media Player, YouTube, Navegador Web, dentre outras.

Após o lançamento da primeira versão, o desenvolvimento contínuo, imposto pelos projetistas do sistema, possibilitou, o surgimento rápido de versões mais robustas do sistema, abarcando diversas novidades, tais como: a introdução de um teclado virtual, o suporte para telas grandes (como tablets), o suporte para dispositivos de jogos e para televisões, algumas melhorias em serviços da Google, entre outras (TANENBAUM; BOS, 2016).

2.2 Histórico de Versões

As principais versões do Android, até então, são: 1.0 ou Alpha; 1.1 ou Beta; 1.5 ou Cupcake; 1.6 ou Donut; 2.0/2.1 ou Eclair; 2.2 ou Froyo; 2.3 ou Gingerbread; 3.x ou Honeycomb; 4.0 ou Ice Cream Sandwich; 4.1/4.2/4.3 ou Jelly Bean; 4.4 ou KitKat; 5.0/5.1 ou Lollipop; 6.0 ou Marshmallow; 7.0/7.1 ou Nougat; 8.0/8.1 ou Oreo; 9.0 ou Pie; e 10 ou Q. (ANDROID OPEN SOURCE PROJECT, 2021; KRAJCI, CUMMINGS, 2013).

2.2.1 Android 1.0

O Android 1.0 ou Alpha, como já relatado no tópico da história, consiste na primeira versão comercial do sistema, lançado para o HTC Dream (T-Mobile G1) em 2008, e trouxe diversas funções e serviços essenciais do sistema operacional, como núcleo baseado no Linux 2.6 com algumas otimizações, sincronização de dados com aplicativos (como Google Contatos), organização de aplicativos, suporte a SMS, presença de notificações do sistema e do usuário, dentre outros (KRAJCI, CUMMINGS, 2013; BROWN, 2018). Além disso, de acordo com Krajci e Cummings (2013), algumas aplicações como Gmail, Google Maps (com StreetView), Media Player, YouTube e Navegador Web foram incorporadas à primeira versão comercial do sistema.

2.2.2 Android 1.1

O Android 1.1 ou Beta trouxe novas funcionalidades importantes a versão anterior, como: identificação de compatibilidade de aplicativo com a versão do sistema (por veio da verificação da API mínima), suporte para letreiros em layouts, suporte para salvar anexos de SMS, correções para a suspensão do dispositivo (sleep), dentre outras novidades (ANDROID DEVELOPERS, 2009). Além disso, segundo a Android Developers (2011), essa versão foi disponibilizada somente para o HTC Dream.

2.2.3 Android 1.5

O Android 1.5 ou Cupcake foi o grande marco do sistema operacional com lançamento em 2009, uma vez que foi a primeira versão a ser lançada com nome de um doce (ideia seguida para as próximas versões) (KRAJCI, CUMMINGS, 2013). Além disso, segundo Krajci e Cummings (2013) e Android Developers (2009), o sistema trouxe como base o núcleo do Linux na versão 2.6.27, um próprio teclado virtual e suporte para teclados de terceiros, novas linguagens de interface do usuário (a fim de se lançar mundialmente), suporte para widgets na tela inicial, gravação e reprodução de vídeos, suporte para Bluetooth, animações durante a inicialização do sistema (boot customizável), dentre outras inovações.

2.2.4 Android 1.6

O Android 1.6 ou Donut, lançado pouco meses após a versão Cupcake, trouxe uma atualização do kernel Linux para a versão 2.6.29 e suporte para o uso de "texto-para-fala" (ANDROID DEVELOPERS, 2009; KRAJCI, CUMMINGS, 2013). Outrossim, alguns recursos novos como: suporte para resoluções de telas do tipo WVGA, melhoras na funcionalidade da câmera, suporte a aplicativos com reconhecimento de gestos, nova barra de pesquisa rápida e indicador do uso de bateria, suporte a Rede Virtual Privada (VPN) e ao Acesso Múltiplo por Divisão de Código (CDMA); também foram introduzidos nessa versão que marcara a última das versões 1.x (ANDROID DEVELOPERS, 2009; KRAJCI, CUMMINGS, 2013).

2.2.5 Android 2.0/2.1

O Android 2.0/2.1 ou Eclair marca a primeira versão da geração 2.x do sistema operacional e manteve a utilização do núcleo do Linux na versão 2.6.29, já que foi lançado em outubro de 2009 (KRAJCI; CUMMINGS, 2013). As mudanças notáveis da versão comercial, segundo o Android Developers (2009), incluem: teclado com dicionário mais inteligente limpeza automática de SMS, navegador web compatível com HTML5, suporte à sincronização de múltiplas contas, suporte a Bluetooth 2.1 e suporte a câmera com flash e zoom digital. Além disso, de acordo com Krajci e Cummings (2013), o principal dispositivo comercializado com essa versão foi o Motorola Droid promovido pela empresa Verizon Wireless.

2.2.6 Android 2.2

O Android 2.2 ou Froyo trouxe, em 2010, as seguintes mudanças significativas: uso da versão 2.6.32 do kernel do Linux, suporte para Adobe Flash e otimização de performance por meio da utilização do Dalvik "just-in-time" (ANDROID DEVELOPER, 2010; KRAJCI, CUMMINGS, 2013). Outrossim, segundo o Android Developers (2010), outras novidades são:

- tela inicial dividida em cinco painéis, com novos widgets em cada painel;
- surgimento dos wallpapers dinâmicos;
- surgimento do Wi-Fi hotspot que, em outras palavras, simbolizava um ponto de acesso portável (por meio do 3G) e que possibilitava o compartilhamento de rede Wi-Fi por até 8 dispositivos;
- múltiplas linguagens de entrada no teclado virtual;
- novo serviço de sincronização na nuvem (Android Cloud to Device Messaging) que possibilitava a sincronização de dados dos aplicativos para os servidores da Google;
- nova ferramenta para os desenvolvedores que possibilitava o relatório dos usuários com relação a erros e problemas para cada aplicativo do Android Market;

- possiblidade de instalação de aplicativos em uma memória externa compartilhada (como um cartão SD), dentre outras inovações.

Por fim, de acordo com Krajci e Cummings (2013), o primeiro dispositivo comercial a ter essa versão foi o Google Nexus One.

2.2.7 Android 2.3

O Android 2.3 ou Gingerbread teve o seu lançamento em 2010 e trouxe grandes mudanças para a plataforma, como: suporte para telas extra largas, do tipo WXGA; suporte para sensores internos de movimento, como giroscópios, acelerômetros e barômetros; suporte para múltiplas câmeras, possibilitando a existência da câmera frontal em conjunto com a câmera traseira e; suporte para a tecnologia de Comunicação Por Campo de Proximidade (NFC) (KRAJCI; CUMMINGS, 2013).

Ademais, de acordo com o Android Developers (2011) e Krajci e Cummings (2013), o Android Gingerbread veio baseado no kernel Linux 2.6.35 e trouxe:

- o suporte para o chamado Open Acecessory, cuja função está associada a dar compatibilidade e gerenciamento para dispositivos USB exteriores (como pen-drives);
- refinamento da interface do usuário, para fins de velocidade e simplicidade;
- teclado virtual com a função de correção e sugestão de palavras;
- teclado virtual com a função de correção e sugestão de palavras;
- novo gerenciador de aplicativos e de memória que possibilitava ao usuário visualizar e gerenciar o consumo de memória e armazenamento;
- novo coletor de lixo (Garbage Collector) do Dalvik, possibilitando o melhor desempenho do sistema;
- novas API abertas da Khronos para áudio (OpenSL) e gráficos (OpenGL).

Por fim, o sistema operacional foi introduzido comercialmente, segundo Krajci e Cummings (2013), ao Google Nexus S.

2.2.8 Android 3.x

O Android 3.x ou Honeycomb, lançado em fevereiro de 2011, foi caracterizado por ser a primeira versão exclusiva e otimizada para dispositivos com tela grande (tablets), tendo a sua primeira aparição no Motorola Xoom (KRAJCI, CUMMINGS, 2013). Por conta disso, segundo Android Developers (2019) e Krajci e Cummings (2013), as características da versão incluem:

- nova interface do usuário, com o surgimento da chamada Barra de Sistema (System Bar), para acesso rápido ao status e notificação do dispositivo, e a Barra de Ação (Action Bar), para controle da aplicação (por meio da seleção de temas, navegação, widgets e outros tipos de conteúdo);
- existência de cinco telas iniciais totalmente customizáveis (atalhos de aplicativos, widgets e novos fundos de tela) e com acesso de qualquer parte do sistema em qualquer contexto;
- introdução a função de multitarefas (Aplicativos Recentes), presente na Barra de Sistema, que possibilitava ao usuário verificar os processos abertos e rapidamente ir de uma aplicação para outra ou fechá-la;
- novas modificações no teclado virtual como acesso ao caracteres especiais e troca de texto para voz na entrada de texto (função presenta na Barra de Sistema);
- novas ferramentas para seleção de texto, possibilitando a cópia, compartilhamento, colar ou procurar na web por meio da Barra de Ação;
- suporte para o Protocolo de Transferência de Mídia ou Foto (MTP e PTP), possibilitando a rápida transferência e sincronização de arquivos por meio de uma conexão USB a uma câmera ou um computador (sem a necessidade de montar o dispositivo de armazenamento em massa USB);
- suporte para conexões de teclados externos ou controladores de jogos, por meio da USB ou Bluetooth;
- atualização dos aplicativos base do sistema, como navegador (possibilitando o modo anônimo, múltiplas abas, sincronização com a conta Google, suporte a JavaScript, dentre outras funções), câmera e galeria (acesso rápido para foco, flash, zoom, câmera frontal, visualização de álbuns em tela cheia, dentre outras funções), contatos e email;
- surgimento dos Fragmentos (Activity Fragments) que possibilita que uma Atividade (maneira como as atividades são lançadas e reunidas) seja dividida em vários subcomponentes, garantindo a existência de interações entre painéis e animações internas bem como a maior eficiência de desenvolver as aplicações;
- novos widgets redesenhados, como a caixa de pesquisa, calendário, data e hora, dentre outros;
- nova framework de animação, possibilitando a mudança de cor, looping em imagens animadas ou até mesmo a existência de animação, para os elementos da Interface do Usuário, como Views, Widgets, Fragments, dentre outros;
- suporte para aceleração de hardware 2D OpenGL, possibilitando uma alta performance na renderização de gráficos 2D;

- novo runtime (Renderscript) para a construção e maior performance para efeitos e gráficos 3D;
- novas mudanças no Dalvik VM para o suporte a processadores multi-core, possibilitando a multi-thread e melhorando significantemente o desempenho do sistema;
- nova compatibilidade de aplicativos das versões anteriores para as telas largas.

2.2.9 Android 4.0

O Android 4.0 ou Ice Cream Sandwich (ICS) foi a primeira versão baseada no kernel do Linux versão 3.0.1, tendo o seu lançamento em outubro de 2011 para o dispositivo Samsung Galaxy Nexus (KRAJCI, CUMMINGS, 2013). A principal novidade desta versão foi fundir a versão anterior Honeycomb com a versão Gingerbread, dando suporte integral para smartphones e tablets o qual foi seguido pelas próximas versões (KRAJCI, CUMMINGS, 2013).

Ademais, conforme Android Developers (2020), as características do ICS incluem:

- evolução da Interface de Usuário por meio do refinamento das animações, tornando as ações comuns mais visíveis e permitindo a navegações por meio de gestos mais intuitivos;
- utilização do layout sobreposto com três botões virtuais na parte inferior, sendo eles o de Voltar (Back), o da Tela Inicial (Home) e o de Aplicativos Recentes (Recent Apps), e da Barra de Sistema na parte superior, possibilitando a visualização de notificações e status do sistema em qualquer contexto de aplicativo;
- novas funcionalidades do gerenciador de multitarefas, mostrando uma imagem recente do contexto dos aplicativos utilizados;
- suporte para criação de pastas e existência de uma aba inferior com aplicativos favoritos, o qual era possível ser visualizado em qualquer uma das cinco telas iniciais;
- suporte para redimensionamento de widgets, possibilitando uma maior interatividade;
- novas ações na tela de bloqueio, como visualização das notificações ou abrir diretamente o aplicativo da câmera;
- surgimento das ações rápidas para as chamadas recebidas, possibilitando enviar mensagens genéricas por algum aplicativo ou SMS, aceitar ou recusar a ligação;
- intuitividade nas notificações, tarefas e abas, possibilitando descartar por meio de um deslize na tela;
- melhoras nas entrada de texto por meio da verificação ortográfica e lista de sugestões;

- novo motor de entrada de áudio para texto, possibilitando os usuários falar continuamente por um tempo prolongado e corrigir rapidamente o texto final (por meio da lista de sugestões);
- novo gerenciador de dados de internet, possibilitando o usuário visualizar e controlar o gasto de internet (mobile ou Wi-Fi);
- novas funções de acessibilidade, como a exploração-pelo-toque que faz o sistema informar por áudio o nome do componente e a sua ação;
- novas ferramentas de compartilhamento e comunicação, possibilitando adicionar informações das redes sociais nos contatos;
- novo calendário unificado que permitia a visualização e gerenciamento de múltiplos eventos;
- novas funções do software da câmera, como foco continuo, exposição zero do obturador, estabilização do zoom de imagens, obtenção de fotos instantâneas durante a gravação de vídeo, existência do foco automático (por meio da detecção de face) e do foco manual (por meio de um toque simples na tela), novo modo panorâmico, novas ferramentas para efeitos durante a gravação, dentre outras novidades;
- aplicativo da galeria redesenhado para agrupar por meio dos álbuns, além da introdução a um editor de foto que tinha as funções de cortar e rotacionar fotos, remover olhos vermelhos, adicionar efeitos visuais, dentre outras;
- surgimento de um widget da galeria que possibilita os usuários olharem as fotos diretamente da tela inicial;
- suporte para captura de tela instantânea;
- introdução as ferramentas para conexão e armazenamento em nuvem, como favoritos no navegador Google Chrome (sincronização direta com as versões de computador) e integração no e-mail (por meio de pastas);
- surgimento do Android Beam, uma ferramenta capaz de compartilhar dados entre dois dispositivos utilizando o NFC;
- desbloqueio de tela por meio do reconhecimento de face, aumentando a segurança; suporte para Wi-Fi P2P (conexão direta de Wi-Fi) e Bluetooth Health Device Profile (voltado para dispositivos médicos e fitness, como medidor de pressão e balanças);
- novas APIs (Interface de Programação de Aplicativos) para os desenvolvedores, como Social, Calendar, Visual Voicemail;
- novo tipo de layout, chamado GridLayout;

- suporte para entrada de caneta (stylus input);
- suporte para VPN.

2.2.10 Android 4.1/4.2/4.3

O Android 4.1/4.2/4.3 ou Jelly Bean consiste numa série de três versões, tendo a primeira lançada em julho de 2012, a segunda lançada em novembro de 2012 e a terceira lançada em julho de 2013, cada qual com suas características e novidades importantes (KRAJCI, CUMMINGS, 2013). A versão 4.1 incorporou o núcleo 3.1.10 do Linux e foi lançada comercialmente para o tablet Asus Nexus 7 (KRAJCI, CUMMINGS, 2013).

Android Developers (2020) afirma que as características e novidades da versão são:

- extensão do tempo de vsync para todos os desenhos e animações, garantindo uma estabilidade e sincronização na taxa de quadros do sistema (evitando latência);
- existência do buffer triplo para uma melhor renderização de gráficos;
- aumento da entrada da CPU nos períodos de inatividade, a fim de se evitar a latência;
- nova ferramenta de coleta de dados do funcionamento do sistema denominada de systrace;
- aprimoramento da acessibilidade por meio do sistema Talkback (leitor de tela) e reconhecimento de toque;
- compatibilidade de texto bidirecional para línguas índicos e árabes, possibilitando a leitura da esquerda para direita ou direita para esquerda;
- suporte a mapa de teclados instaláveis pelo usuário, para a utilização de mais de um tipo de teclado;
- redimensionamento automático de widgets a fim de se otimizar o conteúdo do componente;
- surgimento de animações na inicialização de alguma Atividade (por meio da classe ActivityOptions);
- transição para modo noturno ou tela cheia, em que o primeiro escurece os componentes de barra e o segundo oculta os componentes de barra (barra de status, ação e navegação);
- facilidade na instalação de plano de fundos interativos (pela pré-visualização);
- novas notificações expansíveis (detalhamento) e possibilidade de executar ações nas mesmas;
- possibilidade de armazenar fotos de contato com resolução 720 x 720;

- notificações para novos dispositivos de entrada conectados, como teclado externo ou controlador de jogo (por USB ou Bluetooth);
- aperfeiçoamento do Android Beam;
- compatibilidade com conexão a serviços baseado em DNS multicast, como impressores e câmeras;
- acesso a codecs de mídia de baixo nível;
- compatibilidade com saída de áudio USB;
- surgimento da função de acionar e gravar áudio; compatibilidade com áudio multicanal e porta HDMI;
- novos efeitos no pré-processamento de áudio, como supressão de ruído e controle de ganho automático;
- compatibilidade para executar arquivos de áudio sem pausas;
- aperfeiçoamento do Renderscript;
- surgimento do Google Play Services, o qual tem o objetivo de integrar todos os serviços da Google (como autenticação);
- existência da atualização "inteligente", ou seja, aquela em que somente é atualizado o que de fato mudou de uma versão da aplicação para a outra mais recente.

Já a versão 4.2, de acordo com o Android Developers (2020) e Krajci e Cummings (2013), veio baseada no kernel 3.4.0 do Linux e trouxe as seguintes novidades:

- melhorias na renderização 2D de aceleração por hardware, tornando as animações comuns mais rápidas e interativas (desenhos e páginas de internet);
- melhoras no desempenho de processamento gráfico e de imagens por meio da plataforma Renderscript Compute que possibilitava, sobretudo, o uso automático e otimizado de recursos de computação da GPU dedicada do dispositivo móvel;
- atualização da Interface de Usuário para todos os tamanhos de tela, criando o padrão de barra de status na parte superior (com acesso as notificações e novo menu de configurações rápidas) e da barra do sistema na parte inferior (com os botões virtuais);
- suporte a multiusuário eficiente, não necessitando de mudanças nas aplicações para serem executados em cada ambiente separado (criação de novas instâncias para cada usuário);
- suporte para widgets de aplicativos nas telas de bloqueio;

- novo modo de proteção de tela interativo, denominado Daydream, em que é combinado os melhores recursos de plano de fundo interativos, widgets da tela inicial e notificações;
- compatibilidade a telas externas, por meio do novo serviço de gerenciador de telas que trouxe a integração dos aplicativos a diversas telas e a nova função de seleção de tela preferencial;
- novo componente de interface, denominado Presentation, representando uma janela para conteúdo especificamente a uma tela externa;
- suporte a displays de conexão remota, em conformidade com o programa de certificação Miracast;
- compatibilidade nativa e completa para layouts com leitura orientada da direita para esquerda, possibilitando a adequação de texto conforme o script de idioma;
- otimização de fontes e caracteres, principalmente os relacionados ao oriente (árabes, tailandeses, hebraicos, dentre outros);
- melhorias no teclado virtual com a atualização de dicionários, incluindo adições para diversas línguas como grego, finlandês, polonês, sérvio, sueco, truco, dentre outras;
- suporte para hierarquia do componente de interface denominado Fragments;
- melhorias na acessibilidade, por meio da diferenciação entre a exploração por toque dos gestos de acessibilidade;
- melhorias no software da câmera com o novo modo de cena HDR;
- novas opções para os desenvolvedores, como o modo de depuração, obtenção de relatório de bugs, verificação de aplicativos pela USB, atualização das camadas de hardware, simulação de telas secundárias, rastreamento da execução do OpenGL, dentre outras funções;
- melhorias na segurança, como verificação e filtração dos aplicativos antes da instalação, VPN sempre ativa, fixação de certificados, modificação dos métodos de criptografia, dentre outros;
- melhoras na reprodução de áudio de baixa latência por meio do OpenSL ES;
- melhorias do ambiente Dalvik: compatibilidade com x86 JIT da Intel e MIPS JIT do MIPS, otimização da coleta de lixo;
- compatibilidade intrínseca com alguns métodos da biblioteca StrictMath (funções abs, min, max e sqrt), dentre outras.

Por fim, Android Developers (2020) e Krajci e Cummings (2013) afirma que a versão 4.3 do sistema incluiu as seguintes características:

- melhorias no renderizador 2D como otimização do fluxo de comandos de desenho, suporte a multithreading em diversos núcleos de CPU, maior qualidade de renderização de formas e texto, dentre outras;
- utilização do Khronos OpenGL ES v3.0 para otimização do gerenciamento de texturas e melhoras na renderização de gradientes;
- suporte para a tecnologia Bluetooh Smart que, em outras palavras, garante a interação dos dispositivos móveis com os dispositivos e sensores pequenos, como os relógios inteligentes (smartwatch) e localizadores;
- extensão do recurso de multiusuário para tablets com a introdução dos perfis restritos, garantindo a configuração de ambientes separados com restrições de aplicativos e dados mais refinadas para cada usuário;
- otimização dos recursos de localização e sensores, como o novo modo de somente busca por Wi-Fi (melhorar a precisão da localização e economizar bateria), o novo gerenciamento de leitura dos sensores pelos aplicativos por meio do vetor de rotação (sem preocupações com interferências magnéticas), otimização da energia por meio da fronteira geográfica virtual de hardware, dentre outros;
- melhorias nos recursos de mídia, por meio da nova estrutura DRM modular, da compatibilidade ao codificador VP8 e do multiplexador de mídia (áudio e vídeo MPEG-4);
- novos controles de permissão dos aplicativos a exibirem notificações;
- novo controle sobre a orientação de tela, permitindo os aplicativos bloquearem a tela durante a sua execução;
- melhorias na Interface de Usuário para as línguas de leitura da direita para esquerda, adaptando o sistema e aplicativos para tal;
- melhorias na segurança, por meio do suporte as credenciais de Wi-Fi, utilização do SELinux (sistema de controle do kernel Linux), ferramentas para verificação e armazenamento de chaves de uso exclusivo de aplicativos (para validação), dentre outras;
- melhorias para o desenvolvedor, consistidas em: aprimoramento da ferramenta Systrace (possibilita a coleta de dados dos módulos de hardware, funções do kernel, da máquina virtual Dalvik, dentre outras) e nova ferramenta de criação de perfil de gráficos (linha ou barra) mostrando o consumo de renderização da GPU.

2.2.11 Android 4.4

O Android 4.4 ou KitKat, lançado em setembro de 2013 para os desenvolvedores e comercialmente no Nexus 5 em outubro do mesmo ano, trouxe um importante marco com relação ao desempenho do sistema como um todo, haja visto a inclusão de otimizações de performance para dispositivos com baixa quantidade de memória RAM e a mudança na máquina virtual com a troca do Dalvik para o chamado ART. (KRAJCI, CUMMINGS, 2013).

Outrossim, de acordo com o Android Developers (2020), as principais novidades com relação à versão anterior foram:

- sistema desenvolvido para funcionar em aparelhos com no mínimo 512 MB de memória RAM;
- ajuste de cache do código JIT, combinação de páginas iguais do kernel, troca para zRAM e outras otimizações para gerenciamento de memória;
- inicialização dos serviços em séries, a fim de evitar um consumo excessivo de memória;
- nova API para fornecimento de aplicativos eficientes e responsivos em todos os dispositivos;
- novos recursos da NFC por meio da emulação de cartão host (HCE), em que é possível realizar pagamentos e acesso por cartão;
- suporte nativo para qualquer tipo de impressão por meio de serviços Wi-Fi ou os hospedados em nuvem (Google Cloud Print);
- nova estrutura de acesso ao armazenamento, simplificando a busca de arquivos locais (por aplicação) ou os localizados na nuvem;
- suporte para o agrupamento de sensores de hardware, a fim de reduzir eficientemente o consumo contínuo de energia;
- nova compatibilidade para os novos sensores de detecção e contador de passos;
- novo provedor de SMS compartilhado e novas APIs para o gerenciamento de mensagens;
- novo modo imersivo de tela cheia, escondendo a interface do sistema e possibilitando o uso completo do display do dispositivo;
- novas estruturas de transições de cenas de animação, possibilitando animar mudanças em tempo real;
- nova interface do usuário baseada no modo translúcido do sistema, ou seja, as barras de navegação e status podem exibir o conteúdo da aplicação corrente em seus preenchimentos;

- novos tipos de notificações, com existência de imagens, textos, data e hora, dentre outros componentes;
- nova WebView baseada no Chromium WebView, ampliando a compatibilidade com HTML5, CSS3 e JavaScript;
- novo utilitário de gravação de tela;
- alternância de resolução durante a reprodução;
- novo suporte com o encapsulamento de áudio em um processador de sinal digital (DSP), o que ocasiona a economia drástica da bateria;
- novas ferramentas para monitoramento de áudio;
- novo atenuador de ruído;
- melhorias no desempenho contínuo do RenderScript;
- novo suporte para a aceleração de GPU pelo RenderScript, aumentando o desempenho do sistema;
- possibilidade do uso do RenderScript em estruturas de código nativo;
- novo suporte para os sensores infravermelhos;
- novidades na acessibilidade, como na existência de legendas gerais do sistema;
- melhoras na segurança, como o modo de imposição do SELinux e compatibilidade a dois novos algoritmos de criptografia;
- novas ferramentas para análise do uso de memória e de processos, como a procstats.

2.2.12 Android 5.0/5.1

O Android 5.0/5.1 ou Lollipop, sendo considerado uma das versões mais ambiciosas do Android, trouxe as seguintes novidades, de acordo com o Android Developers (2020):

- novas ferramentas para a interface do usuário por meio de um novo design, denominado Material Design;
- suporte tanto para visualizações em 3D nas interfaces do usuário, como no lançamento de sombras em tempo real para os componentes;
- exclusividade de execução das aplicações pelo ART, por meio dos métodos de ahead-of-time (AOT), just-in-time (JIT) e interpretado;

- compatibilidade para arquiteturas de 64 bits, especialmente para aparelhos focados em jogos;
- modificações nas notificações, possibilitando suas aparições tanto na tela de bloqueio quanto na nova janela flutuante (denominada notificação de informações básicas);
- suporte integral para o chamado Android TV;
- centralização dos aplicativos na função de "Visão Geral" (antigamente denominada de "Recentes"), facilitando o controle do usuário;
- novas APIs para uma conectividade avançada, por meio da utilização do Bluetooth Low Energy (BLE) e dos novos recursos de multirrede;
- compatibilidade com o Khronos OpernGL ES 3.1, oferecendo um alto nível para capacidade gráfica 2D e 3D;
- suporte para entrada de áudio de baixa latência e mixagem de transmissão de áudio multicanal;
- nova compatibilidade com periféricos de áudio USB;
- novas APIs de câmera que possibilitam na captura em formatos raw e melhores nas ferramentas de edição;
- nova codificação de vídeo de alta eficiência (HEVC) do H.265;
- novos recursos de captura e compartilhamento de tela;
- novos suportes para o sensor de detecção de inclinação e um sensor de atividade cardíaca;
- melhorias no Chromium WebView, adicionando compatibilidade com WebRTC, WebAudio e WebGL;
- novas APIs de agendamento de tarefas, as quais permitem a otimização da vida útil da bateria no adiamento de execução de tarefas do sistema.

2.2.13 Android 6.0

O Android 6.0 ou Marshmallow trouxe as seguintes principais mudanças, de acordo com o Android Developers (2020): novo modelo de gerenciamento de permissões do aplicativo em tempo de execução, possibilitando um controle aprimorado sobre as permissões; novas otimizações com dois novos modos, o primeiro sendo chamado de "Soneca" e que mantém o sistema em estado de suspensão, e o segundo sendo chamado de "App em Espera" em que o sistema bloqueia uma aplicação ociosa por muito tempo; novas ferramentas de seleção de texto, por meio de uma barra flutuante em que são exprimido as ações (como recortar, copiar

e colar); mudança no serviço da câmera, trocando o modelo de "primeiro a chegar, primeiro a ser atendido" para o modelo de priorização; validações mais estritas aos pacotes; novas funcionalidades na conexão do dispositivo pela USB, necessitando da permissão do usuário para o acesso a essas funcionalidades.

2.2.14 Android 7.0/7.1

Por meio do Android 7.0/7.1 ou Nougat, segundo o Android Developers (2020), a Google introduziu as novidades a seguir:

- novo recurso de multitarefa que garante a compatibilidade com várias janelas na mesma tela;
- novos tipos de notificações, possibilitando funcionalidades como a resposta direta para a aplicação;
- aprimoramento do desempenho por meio do uso do JIT com perfis de código para ART;
- melhorias no modo "Soneca" para movimentações do dispositivo, possibilitando ainda mais economia da bateria por restrições de rede e CPU;
- surgimento do Project Svelte, uma iniciativa para minimizar o uso de RAM pelo sistema por meio da otimização das aplicações em segundo plano;
- nova função de economia de dados móveis; surgimento das "Configurações Rápidas", em que são expostos as principais configurações e ações diretamente da aba de notificações;
- suporte para o bloqueio de números de telefone;
- novas funcionalidades na aplicação de telefone padrão, como a rejeição de chamada ou não inclusão da mesma no registro;
- compatibilidade a múltiplas localidades e idiomas;
- suporte para a versão do OpenGL ES 3.2, possibilitando novas melhorias gráficas;
- suporte a gravação do Android TV;
- melhoras na segurança por meio de novos esquemas de assinatura dos pacotes;
- compatibilidade com aplicações de Realidade Virtual, por meio da oferta de gráficos de baixa latência.

2.2.15 Android 8.0/8.1

O Android 8.0/8.1 ou Oreo trouxe, conforme explicita o Android Developers (2019), as seguintes principais novidades: novo tipo de notificações, do chamado "Canais de Notificação" em que define que todas as notificações precisam ser atribuídas a um canal (como de alarme ou temporizadores); nova estrutura de preenchimento automático de texto, como os relacionados a login e de cartões de crédito; permissão de inicialização das atividades no chamado modo imagem em imagem (PIP), sendo utilizado principalmente para a reprodução de vídeo; suporte para a instalação de fontes de texto terceirizadas; novo suporte para a utilização de fontes como recursos de aplicações; suporte para o dimensionamento automático da caixa de texto e de ícones; suporte aprimorado para várias telas; atualização dos provedores de conteúdo, possibilitando o carregamento de um conjunto de dados grande por página; novo compartilhamento inteligente de dados das aplicações, baseados nas preferências do usuário.

2.2.16 Android 9.0

O Android 9.0 ou Pie, de acordo com o Android Developers (2019), introduziu os seguintes principais recursos e novidades:

- suporte para o protocolo Wi-Fi Round-Trip-Time (RTT) que permite que os aplicativos usem o sistema de localização e posicionamento em ambientes internos;
- suporte a recortes de tela para câmeras e alto-falantes;
- modificações aos "Canais de Notificação", por meio da funcionalidade de bloqueio de grupo ou de novos tipos de intento de transmissão;
- suporte ao acesso simultâneo de duas ou mais câmeras físicas;
- novas classes relacionadas a animação, no que se refere ao desenho e exibição de imagens animadas GIF e WebP;
- suporte integrado ao perfil 2 do VP9 de Alto Alcance Dinâmico (HDR) e compactação de imagens HEIF;
- introdução e melhorias na API de Rede Neurais;
- melhorias na segurança por meio da caixa de diálogo de autenticação biométrica unificada e no novo esquema de assinatura de APK com rodízio de chaves;
- suporte para fixação da rotação de tela, possibilitando o usuário acionar a rotação manualmente;
- melhoras no desempenho por meio da conversão ART antecipada de arquivos no formato Dalvik Executable (DEX), ocupando menos espaço em disco e em RAM.

2.2.17 Android 10

O Android 10 foi construído em torno de três tópicos: inovação com o aprendizado de máquina e suporte para dispositivos com rede 5G; novas ferramentas e recursos para garantir a privacidade e segurança, e; expansão dos controles de bem-estar digital dos usuários. (ANDROID DEVELOPERS, 2019).

Diante disso, o autor afirma que as seguintes principais mudanças foram introduzidas:

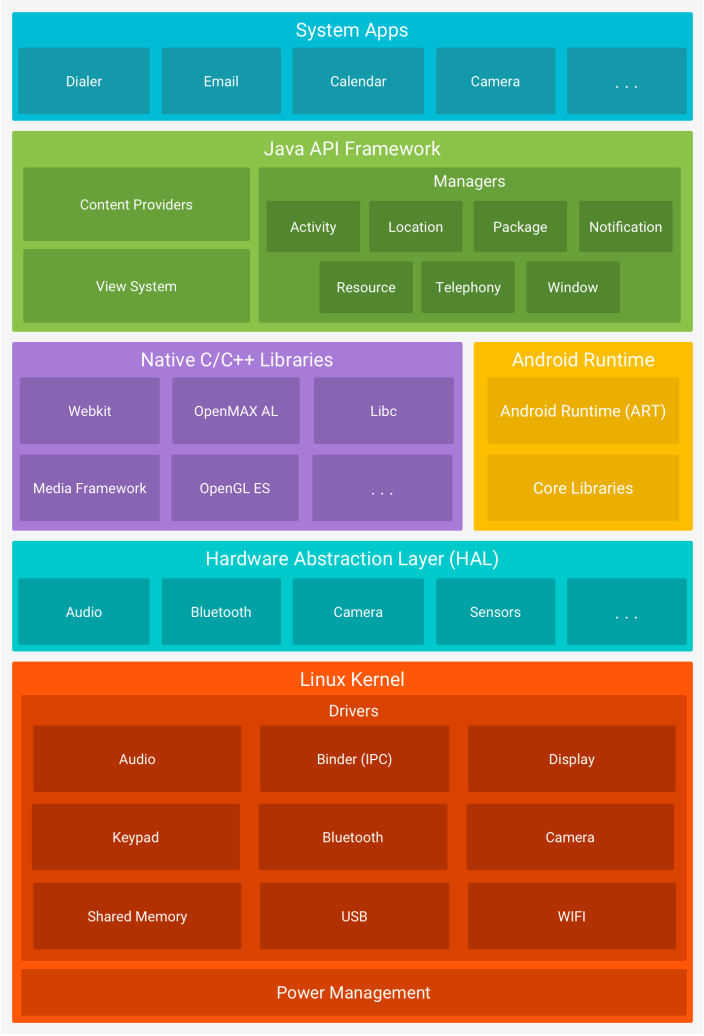
- melhorias no modo de múltiplas janelas, com suporte para dispositivos dobráveis;
- suporte para redes 5G; uso de aprendizado de máquina na sugestão de ações nas notificações, com respostas inteligentes;
- novo tema escuro em todo o sistema;
- novo modo de navegação totalmente por gestos, eliminando a área da barra de navegação e permitindo o uso de tela cheia pelas aplicações;
- novos atalhos de compartilhamento, a fim de que o usuário vá diretamente a outra aplicação compartilhar algum dado;
- maior controle sobre os dados de local, possibilitando o usuário decidir as permissões de localização para determinada aplicação requisitante;
- impedimento de acesso aos identificadores de dispositivo (como IMEI ou número de série) pelas aplicações;
- maior controle do usuário diante dos arquivos de armazenamento externo;
- novo modo de criptografia de dados do usuário, por meio do Adiantum;
- aprimoramento da biometria;
- novo suporte para profundidade dinâmica para fotos, oferencendo opções como desfoque e bokeh;
- novos codecs de aúdio e vídeo, como o AV1 para vídeo e o Opus para áudio;
- facilidade no gerenciamento de dispositivos IoT por meio de funções ponto a ponto;
- otimizações de desempenho com mudanças no ART, por meio da pré-compilação de partes da aplicação.

2.3 Arquitetura da Plataforma

O Android, assim como muitos sistemas operacionais, é composto por um conjunto de softwares em que cada um dos elementos constituintes tem uma certa função e colabora para o funcionamento correto da plataforma. Levando em consideração o exposto, é possível iniciar a discussão em torna da arquitetura do sistema que, segundo Android Developers (2020), pode ser dividida em seis partes: Kernel Linux; Camada de Abstração de Hardware – Hardware Abstraction Layer (HAL), em inglês; Android Runtime; Bibliotecas Nativas C/C++; Framework da Java API; Aplicações.

A figura 1 ilustra a estrutura da plataforma e indica alguns dos elementos presentes em cada uma das camadas.

Figura 1 – Arquitetura do Sistema Android



Fonte: Android Developers, 2020

Nesta seção do trabalho, realiza-se uma síntese das obrigações de cada camada, explicitando o relacionamento de toda a composição do sistema.

2.3.1 Kernel Linux

Segundo Mchoes e Flynn (2013) e Tanenbaum e Bos (2016), o Android foi construído a partir do projeto do kernel Linux, contando com muitos dos recursos já existentes no núcleo em questão para implementar soluções próprias e desempenhar tarefas que são de incumbência de um sistema operacional. Alguns exemplos de recursos aproveitados pelo Android, citados pelos autores, são: o modelo de processos, o algoritmo de escalonamento de processos, o gerenciamento do processador, a gerência da memória, a gerência de memória virtual (swap), o sistema de arquivos, os identificadores de usuário associados a processos e arquivos e os gerenciamentos dos drivers dos dispositivos periféricos e da conexão de rede (MCHOES, FLYNN, 2013; TANENBAUM, BOS, 2016).

É preciso notar que, quando se trata do kernel Linux e do sistema Android, faz-se necessário explanar que o núcleo da plataforma, já instalada em um dispositivo, é composto por quatro conjuntos de softwares: o kernel Linux puro, as adições do AOSP, as correções dos fornecedores de hardware (periféricos) e a customização dos vendedores dos dispositivos móveis (ANDROID OPEN SOURCE PROJECT, 2021).

De modo resumido, vale comentar sobre cada um dos conjuntos de acordo com a visão de Android Open Source Project (2021). O kernel Linux puro é o mesmo núcleo empregado em sistemas para microcomputadores, criado por Linus Torvalds e mantido pela Fundação Linux. As adições do Android, por outro lado, são os mecanismos implementados pelos projetistas da plataforma e ligados ao núcleo puro. As correções dos fornecedores dizem respeitos a drivers e otimizações referentes aos periféricos do dispositivo. E, por último, a personalização dos vendedores consiste em alterações adicionais para dar suporte a funcionalidades do dispositivo em específico.

Neste tópico do presente estudo, discute-se os dois primeiros grupos de software em especial, uma vez que os outros dois conjuntos variam de acordo com o fabricante dos periféricos e com a distribuidora dos dispositivos. Procurou-se focar no que o projeto Android trouxe de novo para o kernel Linux, evidenciado os mecanismos já existentes no kernel e que foram necessários para suportar as extensões.

2.3.1.1 Processos no Android

Android Open Source Project (2020) e Tanenbaum e Bos (2016) conta que o bootloader, software proprietário disponibilizado pelo fornecedor do chip que inicializa o sistema, coloca em execução, inicialmente, o processo init do sistema Android. Os autores afirmam que o init é a raiz dos daemons do núcleo, visto que ele é o encarregado de criar os processos de baixo nível, isto é, processos que envolvem o acesso aos periféricos, o gerenciamento da memória principal, o acesso a arquivos entre outros.

Em segundo lugar, Tanenbaum e Bos (2016) afirmam que o init também traz à tona o

daemon zigoto, ou zygote no inglês, que é a base para todos os processos de alto nível e que contém códigos escritos em linguagem Java. Os autores confirmam que o zigoto é que dá início ao serviço de sistema, ou system_server no inglês, processos que contém os gerenciadores de alto nível, como os gerenciadores de atividades, de pacotes, de energia etc.

De acordo com os autores, o zigoto também é um processo base que serve para dar início a qualquer outro processo de alto nível, como processos de uma aplicação do usuário. Os autores descrevem que quando o gerenciador de tarefas recebe, de algum outro componente do sistema, um pedido para passar uma tarefa a algum processo que ainda não existe, o gerenciador tem que solicitar a criação de um novo processo para o zigoto. Ademais, a criação de novos processos no Android é dada pela replicação do processo zigoto, que já contém um ambiente de execução Java configurado, e pela atribuição de permissões para o processo.

Enfim, segundo Tanenbaum e Bos (2016), o ciclo de vida de um processo é norteado pelo gerenciador de atividades, uma vez que, sendo o responsável por atribuir novas tarefas aos processos, ele é quem estabelece o oom_adj_score do processo, pontuação que indica se o processo pode ou não ser eliminado. Os autores afirmam que quanto maior a pontuação atribuída, mais provável se torna a eliminação do processo, além de que o sistema almeja eliminação dos processos com maior consumo de RAM em primeiro lugar.

2.3.1.2 Sandbox e Permissões

Em consonância com Tanenbaum e Bos (2016), um comportamento comumente encontrado em sistemas operacionais é executar aplicações como se elas fossem o usuário que as iniciou, lógica proveniente da execução de comandos em um terminal. Em outras palavras, os autores afirmam que sistemas geralmente permitem que as aplicações executem com a mesma identidade do usuário, fazendo com que os processos relacionados herdem todas as permissões do usuário.

Executar uma aplicação com as permissões do usuário não seria um problema se o software sempre fosse de confiança, mas, na prática, Tanenbaum e Bos (2016) alegam que o uso de fontes contestáveis pode levar a aplicações que apresentem bugs perigosos ou aplicações mal intencionadas. Os autores ainda lembram que, em alguns sistemas, as aplicações devem solicitar o consentimento do usuário antes de executar com os privilégios do administrador, mas isto não resolve totalmente o problema. Tendo em consideração o prejuízo que uma aplicação com intenções incertas poderia causar gozando de todas as permissões do usuário, os engenheiros responsáveis pelo Android decidiram empregar uma abordagem diferente da usual.

Android Developers (2021) informa que a plataforma Android utiliza o recurso Identificador de Usuário (UID, em inglês) presente no núcleo Linux, para restringir o poder das aplicações, atribuindo permissões individuais a cada processo. Em conformidade com Android Developers (2021) e Tanenbaum e Bos (2016), o sistema atribui um ID de usuário para cada aplicação e faz com que cada aplicação execute como um usuário diferente e com suas próprias

permissões. Por conseguinte, Android Developers (2021) conta que cada processo executa dentro de uma caixa de areia própria, assim uma aplicação maliciosa não pode acessar, por exemplo, arquivos pessoais do usuário, dados de outra aplicação ou áreas do sistema, já que o processo pode ser facilmente detido por insuficiência de privilégios.

No entanto, segundo Android Developers (2020), uma aplicação do usuário, no momento de instalação, pode receber permissões de acesso a recursos fora de sua sandbox, como dados ou ações restritas, mas de modo que não ofereça grande risco ao usuário; essas permissões são enumeradas na página do Google Play sobre a aplicação em questão e/ou na própria tela de instalação. Para mais, o autor informa que algumas aplicações podem solicitar permissões em tempo de execução e essas são as mais perigosas, já que podem conceder acesso a recursos mais restritos, como arquivos pessoais, cabendo ao usuário aceitar ou recusar cada solicitação.

Embora as aplicações possam atuar fora da caixa de areia, ou sandbox no inglês, com a permissão do usuário, elas, em geral, não oferecem perigo ao sistema, uma vez que, segundo Android Developers (2020), a Google Play é uma fonte confiável de aplicativos e, por padrão, o Android impõe uma barreira de segurança a nível do usuário, proibindo a instalação de aplicativos de outras fontes. Vale a pena notar que a barreira pode ser desativada pelo usuário que então passa a assumir certos riscos.

Ademais, a política de segurança gerado pelo mecanismo de sandbox não se restringe aos aplicativos do usuário, mas sim se entende por toda a plataforma Android. De acordo com Android Developers (2021), todas as camadas da arquitetura do sistema, acima do núcleo, estão sujeitas ao modelo da caixa de areia; isto faz com todo o conjunto de software do sistema, seja escrito em código nativo ou em Java, execute dentro de um ambiente isolado e com permissões mínimas.

2.3.1.3 Comunicação entre Processos: ICP do Binder

De acordo com Tanenbaum e Bos (2016), o Android não utiliza os mecanismos de comunicação entre processos originalmente presentes no kernel, tendo em vista que os projetistas optaram por criar um mecanismo próprio da qual eles teriam mais controle e que poderia ser adicionado ao núcleo Linux como uma extensão, o Binder. Salehi, Daryabar e Tadayon (2016) afirmam que o mecanismo está presente além do núcleo, mas também na camada da estrutura da Java API, como a biblioteca libbinder que é carregada diretamente nos processos da plataforma e possibilita, por meio de primitivas, o envio de objetos, de um lado, e o recebimento de objetos, do outro lado.

A biblioteca é capaz de enviar e/ou receber mensagens, por meio de chamadas de sistema, uma vez que o real responsável por controlar a comunicação entre processos é o driver do binder, parte localizada no kernel do sistema (SALEHI; DARYABAR; TADAYON, 2016). Conforme Android Developers (2019) e Salehi, Daryabar e Tadayon (2016) explicam, o binder age como um mensageiro e o seu funcionamento básico pode ser abstraído da seguinte forma:

no processo emissor, a biblioteca ao nível do usuário compacta a mensagem em um pacote, faz uma requisição ao driver do binder (kernel), o driver é que localiza o processo receptor e direciona o pacote, e a biblioteca recebe o pacote, no lado do receptor. Vale ressaltar que qualquer objeto ou tipo primitivo de dado é entendido como mensagem.

Segundo Tanenbaum e Bos (2016), o processo emissor é bloqueado até que a mensagem chegue corretamente ao receptor, que devolve uma confirmação de recebimento; vale notar que os emissores podem evitar o bloqueio, caso não seja desejado. Na verdade, o protocolo de comunicação responsável pela estratégia descrita, conforme os autores afirmam, é a Chamada de Procedimento Remoto, Remote Procedure Call (RPC) no inglês, que consiste em chamar um método no emissor, mas executá-lo no receptor, de forma que qualquer retorno do método seja entregue ao primeiro; é claro que o núcleo precisa fazer o intermédio entre os dois processos para viabilizar a chamada em um e a execução em outro.

Adicionalmente, os autores esclarecem que uma operação de envio de mensagem, denominada transação, pode ser iniciada por um thread qualquer do processo emissor, mas o núcleo encaminha a mensagem para um thread, do receptor, pertencente ao pool de threads, um conjunto de threads que ficam esperando as mensagens. Outro ponto esclarecido por Tanenbaum e Bos (2016) é que o emissor não identifica o receptor quando inicia uma transação, mas sim um determinado objeto presente no emissor, e cabe ao núcleo manter registro dos objetos de cada processo para que seja possível identificar o receptor; no entanto, após identificar o receptor pela primeira vez, o núcleo cria um handle, identificador numérico, que fica associado ao emissor e facilita os próximos envios para o mesmo objeto.

2.3.1.4 Gerenciamento de Memória

A plataforma Android depende do núcleo Linux para atividades como o gerenciamento de memória, mas não utiliza somente os mecanismos existentes no projeto genuíno do kernel, uma vez que os projetistas do Android tiveram que adaptar alguns conceitos e até mesmo criar novos tendo em mente a capacidade limitada de recursos dos dispositivos móveis (MCHOES; FLYNN, 2013). Como exemplo de uma técnica do Linux que foi adaptada para o Android, pode-se citar, segundo Tanenbaum e Bos (2016), o Matador de Falta de Memória, ou Out Of Memory (OOM) Killer do inglês.

De acordo com Android Developers (2019), o Android procura aproveitar ao máximo a memória do dispositivo, inclusive permitindo que alguns aplicativos recém utilizados permaneçam ativos em segundo plano, mesmo que em um estado de menor consumo, o que acaba sendo bom, visto que, por consequência, ajuda no rápido relançamento dos aplicativos. No entanto, para empregar a abordagem descrita, o Android de mecanismos de gerência que saibam lidar com memória baixa.

Android Developers (2019) afirma que a primeira tentativa de combater a memória baixa é feita por um daemon do kernel Linux, o daemon de troca, ou Kernel Swap Daemon

(KSWAPD) no inglês. O autor ainda conta que o mecanismo é ativado pelo núcleo quando a memória livre atinge um limiar preestabelecido, e passa a atuar liberando páginas da memória que não estão sendo utilizadas, como: páginas limpas de memória, que possuem dados copiados do disco e não modificados até então, ou páginas sujas tanto particulares quanto anônimas, ambas em cache.

Em suma, o sistema Android emprega no gerenciamento de memória a técnica de paginação, ou memória virtual, na qual há um endereçamento virtual de alguns blocos (páginas) de memória, visando representar dados ainda no armazenamento secundário (disco) e que podem ser trazidos aos poucos para a memória de acordo com a necessidade dos processos, lógica semelhante à de uma memória cache (ANDROID DEVELOPERS, 2019). Ademais, segundo o autor, as páginas da RAM podem ser classificadas como: em cache, que podem ser particulares ou compartilhadas, além de limpas ou sujas; e anônima, que são sujas.

Uma página de memória em cache contém dados trazidos de algum arquivo do armazenamento secundário e pode ser utilizada por um único processo, configurando uma página particular, ou pode ser compartilhada entre vários processos, configurando uma página compartilhada (ANDROID DEVELOPERS, 2019). Por outro lado, o autor explica que, uma página anônima não é destinada a conter dados que vêm do disco e por este motivo são sempre classificadas como sujas, uma vez que a página limpa é aquela que contém uma cópia idêntica dos dados do disco e a página suja é a que não contém a cópia, mas sim dados alterados ou dados não relacionados ao armazenamento secundário.

Assim, o autor conta que quando o KSWAPD precisa liberar uma página limpa, ele pode simplesmente excluir a página, com a garantia de que nenhum dado será, de fato, perdido; por outro lado, quando o mecanismo precisa liberar páginas sujas, para garantir que os dados não serão perdidos, ele pode mover os dados da página para a zRAM ou, no caso de páginas compartilhadas, para o armazenamento em disco, substituindo os dados originais. A zRAM é uma seção da memória utilizada para as trocas de página, operação na qual a RAM precisa ser desocupada, ou alguns dados precisam ser carregados do disco para a memória ou da memória para o disco; além disso, para não ocupar tanto espaço, os dados enviados para a zRAM são sempre compactados (ANDROID DEVELOPERS, 2019).

Entretanto, algum processo ainda pode querer acessar as páginas que foram excluídas ou movidas para a zRAM e, quando isto ocorre, cabe ao gerente de memória trazer as páginas de voltas, seja copiando os dados do disco para a RAM, seja descompactando os dados da zRAM e trazendo-os de volta para a RAM (ANDROID DEVELOPERS, 2019). É claro que o KSWAPD pode continuar ativo e auxiliar nas trocas de página por meio da liberação de outras, já que o autor deixa claro que o mecanismo só sai de execução quanto um limite aceitável de memória livre é atingido.

Como bem observa Android Developers (2019) e Mchoes e Flynn (2013), o sistema conta com um segundo mecanismo de liberação de memória, que é ativado quando a memória

chega a um limiar crítico e a ação do KSWAPD passa a ser insuficiente, o Matador de Baixa Memória, ou Low-Memory Killer (LMK) do inglês. Mchoes e Flynn (2013) reforçam a ideia do Android Developers (2019), explicando que o sistema Android permite que muitos aplicativos permaneçam executando em segundo plano, alguns desempenhando tarefas que não são essenciais ao usuário no momento e que podem ser eliminados caso em situações de escassez de memória.

Segundo Mchoes e Flynn (2013), para ter uma noção melhor de quais processos podem ser processos eliminados, o gerente de memória emprega a lógica do algoritmo de troca de páginas denominado Menos Recentemente Usado, ou Least Recently Used (LRU), que tem como premissa eliminar as páginas de memória menos acessadas. Outrossim, Tanenbaum e Bos (2016) relaciona o Matador de Baixa Memória do Android com o Matador de Falta de Memória original do Linux, e explica que o princípio comum aos mecanismos é atribuir uma pontuação a cada processo, visando coordenar a eliminação dos processos; a diferença, por outro lado, é que o Android demanda mais eficiência do seu matador, que é empregado frequentemente.

Android Developers (2019) e Tanenbaum e Bos (2016) explicam que a pontuação atribuída aos processos, conhecida por oom_adj_score, considera fatores como a quantidade de acessos recentes, a quantidade de RAM consumida, o quanto de tempo o processo já executou. Ademais, os autores indicam que o aparato começa a matar a partir da aplicação em segundo plano de maior pontuação e continua até que uma quantidade aceitável de memória seja liberada, e, em último caso, o matador pode matar aplicações em primeiro plano e até mesmo serviços de aplicações do usuário, como um serviço de sincronização da conta Google.

2.3.1.5 Memória Compartilhada: ASHMEM e PMEM

Muito embora o sistema Android não suporte a comunicação entre processos por meio de segmentos compartilhados de memória, é possível criar páginas ou segmentos físicos de memória que possam ser utilizados por mais de processo. Segundo Android Developers (2019), aplicações que fazem uso de uma mesma biblioteca ou que precisam de um mesmo serviço podem compartilhar páginas de memória, visando poupar espaços de memória, uma vez que elimina páginas de memória duplicadas. Um exemplo apresentado pelo autor é o serviço de localização, que pode ser compartilhado por aplicações que necessitem desta informação.

De acordo com Brähler (2010) e Drake et al. (2014), o mecanismo (driver) adicionado ao projeto do kernel Linux pelos projetistas do Android para viabilizar esse compartilhamento de memória é denominado Memória Compartilhada do Android, Android Shared Memory (ASHMEM) no inglês. Os autores afirmam que o driver foi projetado para ser compatível com situações de baixa memória e com os mecanismos de liberação de memória.

Ademais, segundo Android Developers (2020), o ASHMEM utiliza uma abordagem baseada em arquivos, na qual cada processo mantém um handle de arquivo enquanto precisar das páginas de memória compartilhadas, alertando o sistema que as páginas devem continuar

mapeadas. Vale ressaltar que o autor apresenta a Java API pela qual processos podem utilizar o driver para a criação e manipulação de blocos partilhados.

Por fim, outro mecanismo de compartilhamento de memória adicionado ao kernel é denominado Memória Física, ou Physical Memory (PMEM) no inglês (DRAKE et al., 2014). Segundo os autores, o driver em questão permite a criação de blocos físicos e contíguos de memória que podem ser utilizados, em conjunto, por processos do usuário e processos ou drivers do núcleo Linux.

2.3.1.6 Gerenciamento de Energia: Wake Locks

Segundo Tanenbaum e Bos (2016), assim como em sistemas operacionais para microcomputadores, o kernel Linux permite que a Unidade Central de Processamento (UCP) entre em um estado de suspensão quando não existem cargas de trabalho importantes ou quando ela está ociosa. No entanto, os autores contam que, quando a UCP adormece, somente uma interrupção externa de hardware, provocada pelo botão de energia do dispositivo, pode despertá-la, o que leva um tempo considerável e não seria interessante para sistemas que têm que estar preparados para tratar, em segundo plano, de vários eventos de natureza frequente.

Serviços de alarme, telefonia, rede entre outros precisam atuar mesmo quando não há processos de usuário em primeiro plano e mesmo que a tela do dispositivo esteja apagada; o mensageiro do sistema, por exemplo, precisa estar apto a receber novas mensagens e a alertar o usuário por meio de notificações, talvez até acendendo a tela por alguns instantes (TANENBAUM; BOS, 2016). Ademais, os autores descrevem que, sempre que necessário, o dono do dispositivo irá ligar a tela para desempenhar alguma tarefa e o tempo de resposta é essencial. Por conseguinte, a suspensão tradicional não se adequa às necessidades da plataforma.

Como os autores bem explicam, manter a UCP ativa ao invés de em um estado de suspensão temporária não é uma boa alternativa, uma vez que a unidade de processamento ficaria sujeita a aceitar qualquer trabalho de baixa importância, impossibilitada de desligar outros dispositivos de hardware e, ainda, por estar ociosa, ficaria sujeita a gastar mais energia do que o normal, característica contraintuitiva presente em alguns equipamentos. Sendo assim, os autores comentam que os projetistas do Android tiveram que elaborar um mecanismo adicional para o kernel Linux, um que viabilizasse a suspensão sem comprometer a usabilidade.

De acordo com Liu et al. (2016), as travas de despertar, ou wake locks no inglês, foram criadas pelos projetistas do Android para que serviços ou aplicações tivessem o poder de conservar a UCP acordada, seja para a execução de certo trabalho ou para o tratamento de determinada condição. Os autores explicam que, para que um processo possa manter o processamento ativo, ele recebe, do núcleo, uma trava que ele pode segurar pelo tempo desejado e depois soltar quando não for mais necessário impedir que o sistema entre em dormência.

Por fim, Tanenbaum e Bos (2016) explana que as travas ficam disponíveis para todo o

sistema e que pelo tempo que o usuário manter a acessa tela, por exemplo, o sistema precisa segurar uma trava para a UCP não adormeça. Em suma, os autores descrevem que quando uma interrupção ocorre, por meio de um alarme por exemplo, a UCP é despertada e o tratador de interrupções passa a segurar uma trava até que algum driver do núcleo assuma o controle para tratar do evento em questão.

2.3.1.7 Outros mecanismos adicionais

Segundo Drake et al. (2014), a plataforma Android criou um aparato de segurança e otimizando dos serviços de rede que é conhecido como Rede Paranoica, ou Paranoid Networking no inglês. O autor afirma que a ideia por trás da tecnologia é criar uma hierarquia, na qual os processos são agrupados de acordo com as suas permissões e a cada grupo, ou nível, é concedido acesso a certas operações relacionadas a rede. Vale ressaltar que por rede entendem-se serviços do Bluetooth, da conexão sem fio, dos dados móveis entre outros presentes em um dispositivo.

Outrossim, outra adição interessante ao kernel do sistema, ainda segundo o autor, foi o logger, um aparato que mantém registros acerca de muitos acontecimentos que possam revelar informações cruciais para a manutenção do sistema. Resumidamente, o sistema registra os ocorridos e os agrupa por meio de quatro categorias: principal, rádio, evento e sistema (DRAKE et al., 2014). Vale dizer que a autor explica que o logger atua juntamente com o mecanismo padrão de registros do Linux.

2.3.2 Camada de Abstração de Hardware (HAL)

A Camada de Abstração de Hardware, ou Hardware Abstraction Layer (HAL), tem como propósito providenciar interfaces para que a estrutura de alto nível do sistema consiga acessar e usufruir dos recursos de hardware (ANDROID DEVELOPERS, 2020). Conforme Android Developers (2020) expõe, a camada em questão contém um conjunto de módulos carregáveis em que cada qual implementa a interface de algum dispositivo; dessa forma, quando uma chamada da Java API é feita, para acessar uma câmera por exemplo, o módulo da câmera é carregado, atuando como um intermediador entre o software (aplicações) e o hardware. A ideia é que o sistema não tenha que se preocupar com a implementação de cada funcionalidade que um dispositivo possa ofertar, mas sim dispor de uma interface específica, isto é, um módulo de biblioteca com rotinas já implementadas pelo próprio fabricante e que pode ser carregado diretamente na aplicação que lhe exigiu.

Ademais, Stoep (2017) conta que a HAL foi repensada, a partir do Android 8, incorporando a criação de um processo para cada módulo carregável e que se comunica com o processo que fez a requisição do recurso por meio do Binder (IPC), melhorando a segurança já que o processo requisitante deixa de ter acesso direto aos drivers do núcleo Linux. Por fim, a partir do Android 9, os processos da HAL obtiveram suporte ao desligamento dinâmico dos

subsistemas de hardware, diminuindo o consumo de memória e de energia (ANDROID OPEN SOURCE PROJECT, 2020).

2.3.3 Android Runtime: ART e Dalvik

Conforme afirma Android Developers (2020), a camada do runtime abriga o ambiente de tempo de execução do Android, conhecido por Android Runtime (ART), que é a máquina virtual encarregada de executar qualquer parte da plataforma que tenha sido escrita em Java. Ademais, a camada em questão também é constituída por um conjunto de bibliotecas de tempo de execução, responsável por muitas funcionalidades da linguagem Java (ANDROID DEVELOPERS, 2020).

Vale mencionar que Android Runtime (ART) é o mecanismo de tempo de execução que passou a ser empregado por padrão a partir do Android 5.0, sendo o sucessor do Dalvik presente nas primeiras versões do sistema (ANDROID DEVELOPERS, 2020). No entanto, os dois mecanismos compartilham o mesmo propósito e possuem um funcionamento semelhante. Além do mais, Android Developers (2020) relata que há uma compatibilidade entre os mecanismos, sendo que aplicativos feitos para Dalvik podem rodar com o ART; o contrário só não é válido, uma vez que o ART traz novidades bem como melhorias ao modelo anterior.

Gunasekera (2012) explica que o Dalvik foi criado para que fosse possível executar aplicações Java em ambientes com recursos reduzidos, como o ambiente encontrado em celulares, que tem que lidar com pouca memória, com uma capacidade de processamento inferior à de microcomputadores e, ainda, com a duração da bateria. Para mais, em consonância com Tanenbaum e Bos (2016), cada componente do Android, escrito em Java, executa dentro de um ambiente Dalvik. Note que as aplicações Java, no Android, ainda são executadas em máquinas virtuais, sendo possível relacionar o Dalvik com a Java Virtual Machine.

Tanenbaum e Bos (2016) explanam que uma aplicação Java convencional, após ser escrita, deve ser convertida para um formato bytecode que é utilizado para a execução; por outro lado, uma aplicação Java para Android, após ser convertida para bytecode, tem que ser convertida para um outro formato intermediário, que é o bytecode do Dalvik, sendo este o formato utilizado para a execução. Outrossim, os autores contam que o formato utilizado pelo interpretador Dalvik é mais compacto que o tradicional tanto em termos de armazenamento quanto em termos de memória RAM.

No entanto, é preciso assinalar que o Dalvik não é responsável por gerenciar as aplicações Java, sequer os serviços de sistema escritos em Java. Conforme os autores frisam, o kernel Linux fica encarregado de implementar o isolamento entre cada componente que executa no Android por meio de processos, de modo que uma aplicação ou serviço em Java, durante a sua execução, tenha tanto um processo vinculado quanto um ambiente de execução Dalvik.

Muito embora a máquina virtual Dalvik seja mais simples do que uma JVM tradicional,

devido ao trabalho conjunto com o Linux, um dos desafios que o uso do Dalvik trouxe foi em termos de velocidade de lançamento das aplicações (TANENBAUM; BOS, 2016). Os autores ainda elucidam que instanciar um ambiente Dalvik para cada processo toma alguns segundos e uma forma de driblar esta barreira foi o emprego do processo zigoto: ao invés de criar um novo processo do zero, o sistema cria uma réplica do zigoto, que tem como função deixar recursos pré-carregados; o processo replicado conta com um ambiente Dalvik pronto para o uso.

Cheng e Buzbee (2010) afirmam que o Dalvik tinha como principal ocupação, além do gerenciamento de memória e coleta de lixo, interpretar o bytecode compacto presente no pacote de instalação de uma aplicação qualquer para Android. Segundo Android Developers (2020), os arquivos que contém o bytecode em questão são denominados Executáveis do Dalvik, ou Dalvik Executable (DEX), e vêm sendo utilizados até o momento vigente pelo novo ambiente de execução, o ART.

Todavia, Cheng e Buzbee (2010) contam que interpretar cada instrução do bytecode, converter para uma instrução equivalente em linguagem de máquina do hardware específico e, por fim, executar a sequência de passos que a instrução necessita é um processo menos eficiente do que executar o código já compilado, fazendo com que aplicações orientadas à UCP tomem muito tempo.

Tendo em vista o problema descrito, os autores afirmam que uma solução inicial proposta pelos projetistas para tornar a execução mais rápida foi disponibilizar, por meio da Java API, um conjunto de métodos estáticos já compilados que os desenvolvedores poderiam utilizar conforme as necessidades de suas aplicações. Porém, como não houve uma melhora considerável, uma outra solução foi elaborada: o uso da técnica de perfilamento de código e de um compilador Just-In-Time (JIT).

Cheng e Buzbee (2010) explicam que o perfilamento consistia em empregar ferramentas de monitoramento durante a execução de uma aplicação qualquer, visando dividir o código da aplicação em: código quente, ou hot code no inglês, e código frio, ou cold code no inglês. Vale esclarecer que, por código quente, entende-se as partes do código que ocupam boa parte do tempo de execução de um aplicativo. Ademais, os autores também explicam que compilação JIT é uma técnica que, em suma, compila o código pouco a pouco, sendo que um método ou uma porção do código só é compilado pouco antes da sua execução acontecer.

Segundo os autores, a solução empregada resume-se em identificar, em tempo de execução, qual é o código quente, que deixa de ser interpretado e passa a ser compilado pelo compilador JIT; por outro lado, a porção de código identificada como fria continua a ser interpretada. Com esta abordagem, aplicativos com intensa computação puderam executar de modo mais eficiente, mas, no geral, houve um aumento do uso da UCP, da memória principal e um gasto maior de energia (GEOFFRAY; JURAVLE; CHARTIER, 2016).

Em consonância com Geoffray, Juravle e Chartier (2016), o ART substituiu o Dalvik no

Android 4.4, codinome KitKat, e trouxe com ele um compilador Ahead-Of-Time (AOT), que tinha como objetivo compilar integralmente todos os arquivos DEX no momento da instalação do aplicativo, como é feito tradicionalmente em outros sistemas operacionais. Os autores afirmam que o novo ambiente de execução surgiu para resolver os problemas de otimização de recursos que o Dalvik estava enfrentando, além de introduzir melhorias tanto na coleta de lixo quanto no gerenciamento de memória de alto nível. Outrossim, os autores informam que, com o ART e o AOT, a UCP, a memória e a bateria tiveram menores índices de consumo, mas o tempo de instalação se tornou superior e o armazenamento do dispositivo se tornou insuficiente para dar conta de tantas aplicações compiladas.

Enfim, a abordagem adotada por último, até o momento vigente, é o uso conjunto do interpretador, do compilador JIT e do compilador AOT (ANDROID DEVELOPERS, 2020). Nesta abordagem, o perfilamento de código é realizado para determinar o que é quente e o que é frio, e, em tempo de execução, código quente passa a ser compilado pelo JIT, enquanto código frio é interpretado. Em um segundo momento, após a execução, o código identificado como quente é compilado pelo AOT para ser executado diretamente da próxima vez.

2.3.4 Bibliotecas Nativas: C/C++

Krajci e Cummings (2013) explicam que a camada em questão é composta pelos binários de diversas bibliotecas escritas em C e C++. Essas bibliotecas têm como encargo, conforme Android Developers (2020) afirma, suportar partes do sistema que sejam escritas em código nativo, como o ART e o HAL, além de possibilitar a execução de aplicações do usuário que contenham código C/C++ ou disponibilizar, por meio do framework da Java API, alguns serviços essenciais ao sistema Android.

Como Android Developers (2020) exemplifica, libc e libc++ são, respectivamente, as bibliotecas padrões do C e do C++ e que contêm rotinas usuais, como as de entrada e saída, sendo necessárias para o suporte de código nativo. Por outro lado, um exemplo de biblioteca que fornece funcionalidades ao sistema é a biblioteca gráfica OpenGL/ES que, segundo Krajci e Cummings (2013), adiciona o suporte à renderização 2D e 3D ao sistema e que pode ser acessada pelo alto nível através da interface Java OpenGL API. Ademais, outras bibliotecas nativas, como Surface Manager (Gerenciador de Janelas para diferentes aplicações), SQLite (banco de dados local) e WebKit (renderização de páginas HTML), também são incluídas no sistema. (KRAJCI, CUMMINGS, 2013).

Diante disso, a Google oferece o Kit de Desenvolvimento Nativo (NDK) que tem como finalidade fornecer, em síntese, um conjunto de ferramentas que possibilitam tanto a utilização das bibliotecas nativas da plataforma quanto a programação de aplicações Android em linguagem C/C++ (ANDROID DEVELOPERS, 2020). Os componentes empregados na depuração e na compilação do código nativo de uma aplicação, segundo Android Developers (2020), são: o NDK, que contém as bibliotecas nativas e habilita o uso de códigos C/C++;

o CMake, uma ferramenta de compilação externa para a criação de bibliotecas nativas; e o LLDB, um depurador de códigos nativos.

Por fim, em consonância com Android Developers (2020), as bibliotecas nativas que o NDK abarca são capazes de gerenciar tanto atividades nativas quanto os componentes físicos do dispositivo no qual o sistema está instalado - sensores ou entradas de toque, por exemplo; e o emprego do código nativo possibilita um maior desempenho do dispositivo e permite a execução de aplicações que exigem um processamento elevado, como alguns jogos.

2.3.5 Framework da Java API

O Framework da Java API constitui a camada que abrange o conjunto completo de recursos do sistema operacional pelas APIs da linguagem Java (ANDROID DEVELOPERS, 2020). API é uma sigla para Application Programming Interface ou Interface de Programação de Aplicação que, por definição, constitui um conjunto de rotinas e protocolos padronizados utilizado no desenvolvimento e integração de algum software, viabilizando a comunicação entre produtos sem saber como foram implementados. (RED HAT, [20–]).

Além disso, essa Framework é o principal componente do Kit de Desenvolvimento de Software (SDK), em que os desenvolvedores usufruem das classes e interfaces Java na construção das aplicações (SARKAR et al., 2019). Esse kit, de acordo com o Android Developers (2020), provém todos os pacotes e recursos necessários para o desenvolvimento de aplicações em Android pela linguagem de programação Java ou, mais recentemente, Kotlin. Esses pacotes incluem: as ferramentas de linha de comando do SDK, tais como avdmanager, retrace, sdkmanager, dentre outros; o Android SDK Build Tools, utilizado para a criação e compilação das aplicações; o Android SDK Platform Tools, o qual possui as ferramentas para cada versão nova da Plataforma Android; e o Android Emulator que consiste, basicamente, de uma ferramenta de emulação baseada em QEMU para a execução em tempo real de alguma versão do sistema Android (ANDROID DEVELOPERS, 2020). Além disso, segundo o Android Developers (2020), o SDK acompanha, opcionalmente, um ambiente de desenvolvimento que integra todos esses pacotes e diversos outros recursos (como depuração), denominado Android Studio.

Os recursos abordados por esta camada consistem em: gerenciamento de pacotes, atividades, serviços, janelas, recursos, notificações, localização e telefonia; provedor de conteúdo e sistema de visualização (ANDROID DEVELOPERS, 2020; ANDROID DEVELOPERS, 2020; SARKAR et al., 2019; GUNASEKERA, 2012). Os principais gerenciadores serão abordados abaixo neste tópico, bem como o provedor de conteúdo e o sistema de visualização.

2.3.5.1 Gerenciador de Pacotes

Um pacote, para o sistema operacional Android, é um arquivo com extensão .apk e agrupa os seguintes componentes das aplicações: um manifesto (Android Manifest) em XML

que descreve a aplicação num geral (o que é, como executar, pacote principal, atividades principais, ponto de entrada da aplicação, etc.); os recursos necessários para a execução da aplicação, tais como layouts, imagens gráficas, sons, dentre outros; o código em si, isto é, as classes em Java que contém as rotinas de funcionamento da aplicação; e um arquivo com informações de assinatura, a fim de garantir a segurança (TANENBAUM; BOS, 2016). Além disso, de acordo com Android Developers (2020) e Tanenbaum e Bos (2016), denomina-se gerenciador de pacote a estrutura que controla todas as operações que envolvem os pacotes, isto é, análise do manifesto, instalação e desinstalação das aplicações, permissão para utilização de recursos (como Bluetooth), dentre outras operações.

2.3.5.2 Gerenciador de Atividades

Uma atividade ou Activity, segundo Android Developers (2020) e Tanenbaum e Bos (2016), consiste em um componente cujo objetivo é servir como ponto de entrada de uma aplicação, ou seja, é ela quem garante a interação de um usuário com o app e é essencial para a navegação, seja no próprio app ou entre apps. No geral, de acordo com Mchoes e Flynn (2013), uma aplicação tem um conjunto de atividades, incluindo aquelas que fazem comunicações com outras aplicações. Além disso, como visto no subtópico anterior, o responsável por organizar tanto qual atividade irá executar inicialmente como outras configurações como as relacionadas a orientação de tela é o manifesto (Android Manifest).

Diante disso, este componente possui funções associadas ao seu funcionamento as quais definem o seu chamado ciclo de vida ou, especificamente, mudanças de estados (ANDROID DEVELOPERS, 2020). Existem atualmente sete rotinas/estados associadas ao ciclo de vida de uma atividade, de acordo com o Android Developers (2020) e Mchoes e Flynn (2013):

- onCreate() ou Estado de Criação: acionado quando o sistema cria a atividade, ou seja, está associado ao estado de alocação de recursos e criação do processo;
- onStart() ou Estado de Inicialização: torna a atividade visível ao usuário ao passo que ela é inserida na pilha de retorno e tendo a sua interação configurada;
- onResume() ou Estado de Funcionamento: representa a execução corrente da atividade até a ocorrência de outro evento, garantindo a interação do usuário e movendo-a para primeiro plano;
- onPause() ou Estado de Pausa: ocorre como uma primeira indicação de alguma interrupção, simbolizando a ação do usuário supostamente "deixar a atividade", ou seja, representa que ela não está mais em primeiro plano. Neste estado, há duas possibilidades: a atividade pode ser retornada imediatamente, o que ocasiona a chamada da rotina onResume(); ou a atividade pode ser trocada por outra, o que ocasiona na chamada da rotina onStop();

- onStop() ou Estado Parado: representa quando a atividade, de fato, não está mais em primeiro plano do usuário, ou seja, não é mais visível. Neste caso, a atividade será bloqueada e deixada na memória, para o caso de retorno do usuário, o que acionaria o método onRestart();
- onRestart() ou Estado Reiniciado: acionado quando a atividade, que está bloqueada, é acionada novamente pelo usuário. Diante disso, este método levará a atividade para o Estado de Inicialização;
- onDestroy() ou Estado de Destruição: acionado um pouco antes do encerramento da atividade. Neste caso, pode ser chamado tanto quando o usuário descarta a atividade por completo quanto o sistema encerra a atividade por alguma mudança de configuração (como rotação de tela ou modo de várias janelas).

Essas rotinas associadas a cada estado de funcionamento do componente são realizadas pelo chamado gerenciador de atividades. Além disso, este gerenciador garante outras funções como: mudanças de orientação ocorrerem de modo suave, não ocorrência de perda de dados do usuário durante a transição entre atividades e eliminar processos quando apropriado (ANDROID DEVELOPERS, 2020).

Conforme Tanenbaum e Bos (2016) explicam, o funcionamento do gerenciador de atividades, com relação ao ciclo de vida da atividade, ocorre da seguinte forma: quando o usuário abre uma aplicação, é identificado o ponto de entrada da mesma (por meio do manifesto) e ocorre uma instanciação (criação) desse componente do tipo Activity e o mesmo é inserido em uma estrutura denominada tarefa (task). Ainda, existe uma estrutura denominada pilha de retorno que, segundo o Android Developers (2020), insere as atividades na ordem em que foram abertas, a fim de fornecer um controle básico de para a transição e encerramento das atividades (este último sendo ativado por meio do botão "Voltar" do dispositivo ou por alguma rotina programada especificamente para isso). A partir disso, o gerente aciona o processo da aplicação e exibe a interface de usuário associada àquela atividade principal, ativando o estado de funcionamento da mesma.

Neste ponto, é necessário definir a respeito da interface de usuário denominada "Recentes" ou "Visão Geral" que basicamente consiste em um acionamento do gerente de tarefas, listando as atividades e tarefas acessadas recentemente e possibilita ao usuário retornar ou remover alguma tarefa da lista (por meio do deslize de tela) (ANDROID DEVELOPERS, 2020).

Diante disso, existem três situações possíveis de saída da aplicação que o usuário pode realizar a partir do estado de funcionamento da atividade: ir a tela inicial do sistema, por meio do botão home do dispositivo; realizar a troca ou remoção, por meio do gerente de tarefas. Nas três situações, conforme explicitam Tanenbaum e Bos (2016), o gerente de atividades irá solicitar ao sistema para salvar o estado (neste caso, trata-se das informações essenciais da

atividade atual) em um objeto do S.O. denominado ActivityRecord e, a partir disso, a atividade é colocada como "parada".

Neste momento, a primeira e segunda situações diferenciam-se da terceira situação pelo seguinte conceito: o sistema, que realiza uma varredura na memória RAM, pode indicar que outro processo ou aplicação necessite de mais memória e, portanto, acaba por excluir o estado salvo anteriormente pelo gerente de atividades, necessitando criar a atividade novamente de forma automatizada, nas primeira e segunda situações, ou de forma manual, na terceira situação, sendo essa última associada ao estado de destruição da aplicação. Por outro lado, o usuário pode voltar para a aplicação após sair, ocasionando o gerente de atividades a retomar o estado salvo e acionar o processo da aplicação novamente.

Por fim, a outra situação que o usuário pode realizar: a do compartilhamento de dados entre atividades. Para tanto, é necessário abordar a respeito de uma outra estrutura importante do sistema operacional Android: o Intento.

O Intento é um objeto que facilita a comunicação entre os componentes em três formas: no início de uma atividade, em que o intento descreve a atividade e carrega todos os dados necessários; no início de um serviço, semelhante ao da atividade; e no fornecimento de uma transmissão (ANDROID DEVELOPERS, 2020). Além disso, existem dois tipos de intentos, segundo Tanenbaum e Bos (2016) e Android Developers (2020): o explícito, o qual especifica qual aplicação usufruirá do intento, fornecendo o nome do pacote da aplicação de destino ou o nome da classe de algum componente e, portanto, possibilita o acionamento de algum componente na própria aplicação (como outra atividade ou serviço); e o implícito, o qual é declarado uma ação geral e possibilita, neste caso, a algum componente de outra aplicação a processar tal ação (como na utilização de um serviço específico).

Dessa forma, é possível que os usuários compartilhem dados entre aplicações diante de duas possibilidades pré-programadas da aplicação: o Android Sharesheet ou o resolver de intentos padrão do S.O. (ANDROID DEVELOPERS, 2020). Tanto o primeiro quanto o segundo, de acordo com o Android Developers (2020), utilizam as funções do objeto Intent, mas oferecem diferentes possibilidades ao usuário: enquanto que o Android Sharesheet tem a capacidade de sugerir todos os destinos que possam comportar o dado a ser carregado, o resolver de intentos padrão mostra somente as aplicações específicas para o transporte daquele tipo de dado. Por exemplo, suponha o compartilhamento de uma foto de extensão .png a partir da aplicação Câmera: no primeiro caso, o sistema ofereceria uma janela tanto com os contatos recentes em aplicações comumente utilizadas (como WhatsApp) quanto todas as aplicações que supostamente suportam o carregamento deste arquivo .png (como e-mail, gerenciador de arquivos, armazenamento em nuvem etc.); já no segundo caso, o desenvolvedor da aplicação deixaria específico o formato .png na utilização do intento, o que ocasionaria o sistema a sugerir a utilização daquelas aplicações que deixaram explícito que suportam o formato .png.

2.3.5.3 Gerenciador de Serviços

Um serviço representa um componente de aplicação que realiza operações longas, contínuas ou não, e não fornece uma interface de usuário (ANDROID DEVELOPERS, 2020). Existem, no sistema operacional Android, três diferentes tipos de acordo com o Android Developers (2020): os de primeiro plano, cuja função está associada a execução de uma operação que é visível ao usuário, como a manipulação de E/S de arquivos; os de segundo plano, cuja função está associada a execução de uma operação que não é perceptível ao usuário, como na reprodução de música; e os vinculados a um componente, os quais estão associados a oferta de uma interface servidor-cliente que permite aos componentes interagirem entre si, mesmo em processos que utilizam o IPC.

Semelhante a uma atividade, um serviço também possui um ciclo de vida, porém muito mais simples e que é gerenciado pelo chamado gerente de serviços. De acordo com o Android Developers (2020), um ciclo de vida de um serviço pode separado em dois caminhos: o primeiro associado a um serviço do tipo iniciado (primeiro ou segundo plano); e o segundo associado a um serviço vinculado.

Sendo assim, segundo o Android Developers (2020), para os serviços iniciados: a primeira etapa ocorre quando um componente chama a rotina startService(), possibilitando a criação do serviço na rotina onCreate() da atividade em questão. Após isso, é acionado a rotina onStartCommand(), a qual ocasiona o início do serviço durante o funcionamento (primeiro plano) ou bloqueio (segundo plano) da atividade. Por fim, o serviço é destruído na mesma chamada de destruição de uma atividade, ou seja, a rotina onDestroy().

Já para os serviços vinculados, ainda de acordo com o Android Developers (2020a), o ciclo de vida ocorre da seguinte maneira: a primeira etapa ocorre a partir da chamada de bindService() por outro componente (cliente), a qual possibilita a criação do serviço no método onCreate() da atividade. Diante disso, é acionado a rotina onBind() que possibilita a comunicação do serviço pelo Binder durante o funcionamento da atividade. Por fim, durante o encerramento da atividade, os vários clientes conectados no serviço acionam a rotina onUnbind(), cuja função está associada ao encerramento da conexão de todos os clientes com o serviço e, dessa forma, o serviço pode ser destruído semelhante ao descrito no parágrafo anterior.

2.3.5.4 Gerenciador de Recursos

Os recursos nada mais são do que arquivos adicionais, geralmente em XML, que representam um conteúdo estático como: bitmaps, interfaces do usuário (layouts), strings associadas a interfaces do usuário, instruções de animação, dentre outros. (ANDROID DEVELOPERS, 2019). Dessa forma, existe um gerenciador específico que coordena o acesso a esses arquivos pela aplicação, denominado gerente de recursos (SARKAR et al., 2019). Além disso, como bem recomenda o Android Developers (2019), os recursos da aplicação devem sempre ser

independentes da aplicação, de tal forma que se torne fácil a sua manutenção e atualização.

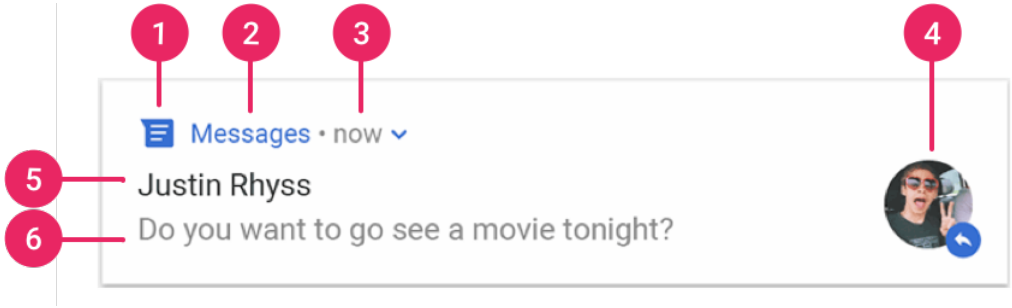
2.3.5.5 Gerenciador de Notificações

Uma notificação nada mais é do que uma mensagem que o sistema exibe fora da interface do usuário da aplicação, a fim de fornecer alertas e comunicados importantes de alguma aplicação. (ANDROID DEVELOPERS, 2021). Dessa forma, como deixa explícito Sarkar et al. (2019), o responsável desde as requisições da aplicação até a construção e exibição da notificação final é o chamado gerenciador de notificações.

Existem diversos locais os quais a notificação pode aparecer: na barra superior de status, com a abertura da chamada "gaveta" de notificações por um deslize para baixo; por uma janela flutuante denominada notificação de informações básicas, a qual aparece por um período de tempo na parte superior do dispositivo; na tela de bloqueio, com a possibilidade de visualizar detalhes ou não; e mais recente (a partir do Android 8.0) com o chamado indicador do ícone da aplicação, em que os usuários podem tocar e manter pressionado no ícone da aplicação requisitante e visualizar as notificações. (ANDROID DEVELOPERS, 2021). Ainda, é importante ressaltar que todas essas formas de ocorrência de notificações possibilitam a utilização de ações, como dispensar e abrir a aplicação.

Por fim, a figura 2 mostra a anatomia de uma uma notificação. De tal forma que:

Figura 2 – Anatomia de uma Notificação



Fonte: Android Developers, 2021

1. Ícone pequeno: representa o ícone da aplicação, em tamanho pequeno no canto superior esquerdo da notificação.

2. Nome da aplicação: representa o nome da aplicação e é fornecido pelo sistema.

3. Carimbo de Data e Hora: é um item opcional, mas representa a data e hora da aparição da notificação, sendo fornecido pelo sistema.

4. Ícone grande: também opcional, sendo geralmente utilizado para foto de contatos.

5. Título: também opcional, representando o título do conteúdo da notificação, como o nome de um contato

6. Texto: também opcional, representando o texto do conteúdo da notificação, como a mensagem de um contato.

2.3.5.6 Gerenciador de Localização

O gerente de localização tem a principal funcionalidade de garantir a privacidade do usuário com relação aos chamados serviços de localização. (ANDROID DEVELOPERS, 2021). Ou seja, está associado ao gerenciamento de solicitações e permissão de acesso as localizações do usuário por alguma aplicação requisitante.

Além disso, como exprime o Android Developers (2020b), o gerente pode processar em dois casos de acesso à localização: as relacionadas com aplicações de primeiro plano, podendo compartilhar ou receber informações da localização apenas uma vez ou por um período definido; ou as relacionadas com aplicações em segundo plano, as quais necessitam do compartilhamento constante da localização ou da utilização da API Geofence - relacionadas às fronteiras geográficas virtuais.

2.3.5.7 Gerenciador de Telefonia

O gerenciador de telefonia está associado com a biblioteca Telecom, a qual gerencia todas as chamadas de áudio e vídeo de um dispositivo, incluindo aquelas com base em chip ou chamadas do tipo VoIP. (ANDROID DEVELOPERS, 2020). Diante disso, o Android Developers (2020) explicita os dois principais componentes da Telecom: o ConnectionService, responsável pela conexão de chamadas a outra parte utilizando algum meio (como VoIP); e o InCallService, responsável pelo fornecimento de uma interface de usuário para as chamadas gerenciadas por essa biblioteca (como a aplicação Telefone). Portanto, é possível tanto criar um substituto para a aplicação padrão de telefonia do S.O., por meio da implementação do componente InCallService, ou para a criação de uma aplicação de chamada que se integre aos recursos de chamada pré-estabelecidos do Android, por meio da implementação do ConnectionService.

2.3.5.8 Provedor de Conteúdo

O chamado provedor de conteúdo é um mecanismo utilizado para o gerenciamento de acesso de dados de uma aplicação, possibilitando principalmente no compartilhamento dessas informações entre diferentes aplicações. (ANDROID DEVELOPERS, 2019; TANENBAUM; BOSS, 2016; SARKAR et al., 2019). Em outras palavras e de acordo com o Android Developers (2019), os provedores de conteúdo nada mais são do que uma interface padrão que conecta dados de um processo em execução para outro processo na memória, possibilitando a modificação desses dados pelas outras aplicações.

De um modo geral, esse mecanismo funciona da seguinte maneira: são estabelecidos URIs que identificam dados de um provedor (autoridade) e o nome que aponta para uma tabela de dados (caminho). (TANENBAUM; BOSS, 2016). A partir disso, o próprio provedor de conteúdo

interpreta a URI e determina quais dados dentro dele estão sendo acessados, possibilitando a interação com outras aplicações por meio de uma chamada do tipo ContentResolver.

2.3.5.9 Sistema de Visualização

O sistema de visualização, como o próprio nome diz, é tudo aquilo que o usuário pode ver e interagir. Neste caso, representa o que é chamado de interface do usuário de uma aplicação. (ANDROID DEVELOPERS, 2021). É neste conceito em que são inseridos, de fato, os componentes relacionados a visualização de uma aplicação, como: botões, caixas de texto, navegador interno, listas, caixas de diálogo, notificações, menus, dentre outros. (ANDROID DEVELOPERS, 2021; SARKAR et al., 2019). Além disso, como bem exprime o Android Developers (2021), os chamados Views são objetos que "desenham" esses componentes e, portanto, são eles quem garantem a visualização tanto dessas estruturas como as relacionadas aos recursos (como layouts).

2.3.6 Aplicações

As aplicações constituem a última parte da arquitetura e representam o contato direto do sistema com o usuário interativo. Em outras palavras, como explicita Sarkar et at. (2019) e Gunasekera (2012), é nesta camada em que os códigos serão compilados e executados por meio do Android Runtime, utilizando de fato os recursos disponíveis do dispositivo. Alguns exemplos de aplicações padrão do sistema são: mensageiro, e-mail, navegador de Internet, calendários, contatos, dentre outras (ANDROID DEVELOPERS, 2020; KRAJCI, CUMMINGS, 2013).

Além disso, conforme explica o Android Developers (2020), as aplicações do sistema fornecem um suporte para os desenvolvedores acessarem os recursos direto de sua aplicação, por meio de uma invocação de serviço. Isso traz uma facilidade ao programador, haja visto que não é necessário programar inteiramente determinado serviço para utilizar suas funções como, por exemplo, na abertura do navegador de Internet quando requerida uma pesquisa dentro de uma certa aplicação. Além disso, como deixa claro a documentação, um aplicativo terceirizado pode substituir algum de sistema por meio da escolha do usuário, o que lhe garante liberdade.

3 Metodologia

O trabalho em questão integra uma revisão bibliográfica a respeito do sistema operacional Android com especial enfoque na arquitetura do conjunto de softwares. Desta forma, foram abordados diferentes tópicos relacionados ao Android, constituindo: uma breve revisão da história e do surgimento do sistema operacional; um histórico das versões e as mudanças entre cada uma, até a versão vigente no ano de realização do trabalho; uma divisão dos principais componentes da arquitetura do Android, além de um detalhamento a respeito das responsabilidades de cada um dentro do funcionamento do sistema.

Para tanto, a monografia está baseada em um caráter de pesquisa bibliográfica que, segundo Pizzani et al. (2012), consiste na realização de um levantamento de informações sobre o tema com base em diversas fontes, como livros, periódicos, artigos de jornais, sites da Internet, dentre outros. Além disso, as autoras ainda expressam que esse tipo de pesquisa tem diversos objetivos, dentre eles o de proporcionar um aprendizado acerca de determinada área do conhecimento (PIZZANI et al., 2012). Por consequência, pode-se afirmar que o presente documento serve como uma fonte histórica e de referência para que entusiastas da computação possam entender como funciona o sistema operacional Android.

Enfim, tendo em mente o exposto, evidencia-se abaixo o processo de coleta e análise de dados que visou proporcionar uma explicação direta e de fácil entendimento sobre os componentes citados anteriormente.

3.1 Coleta e Análise de Dados

O processo de coleta dos dados ocorreu em quatro partes. Primeiramente, foi realizada uma pesquisa por publicações cientificas e/ou livros a respeito do sistema operacional Android nas seguintes bases de dados: IEEE Xplore, ACM, SciELO e CAPES. Dentre as palavras-chaves utilizadas estão “Android”, “Operating System” e “Architecture”; além de que, o período de publicação considerado foi de cinco anos. Segue abaixo uma tabela com o número de resultados encontrados em cada base.

Quadro 1 – Comparativo de resultados em quatro diferentes bases de dados

Base de Dados	**Palavras-chave**	**Período de Publicação**	**Número de Resultados**
IEEE Xplore Digital Library	Android; Operating System; Architecture	Últimos cinco anos: entre 2016 e 2021, inclusivo	255
ACM Digital Library	Android; Operating System; Architecture	Últimos cinco anos: entre 2016 e 2021, inclusivo	4.507
SciELO	Android; Operating System; Architecture	Últimos cinco anos: entre 2016 e 2021, inclusivo	2
Periódicos CAPES	Android; Architecture	Últimos cinco anos: entre 2016 e 2021, inclusivo	6.085

Fonte: Elaborado pelos autores, 2021.

Diante dos resultados obtidos por ordem de relevância, foi feita uma análise das três primeiras páginas de cada base. É possível observar pelos títulos, encontrados nas primeiras páginas, que não houveram artigos com alguma proposta semelhante à do presente trabalho, isto é, não foram encontrados estudos bibliográficos acerca da arquitetura do Android. Entretanto, nas bases IEEE Xplore e ACM, houveram alguns artigos relacionados a componentes internos da arquitetura Android, além de artigos comentando sobre segurança e performance. Nessa fase, dois artigos do IEEE Xplore foram selecionados para a análise posterior, sendo eles:

- SALEHI, Majid; DARYABAR, Farid; TADAYSON, Mohammad Hesam. Welcome to Binder: A kernel level attack model for the Binder in Android operating system. *In*: INTERNATIONAL SYMPOSIUM ON TELECOMMUNICATIONS, IST, 2016, Tehran, Irã. **Proceedings** [...]. [*S. l.*]: IEEE, 2016. p. 156-161. Disponível em: https://ieeexplore.ieee.org/document/7881801. Acesso em: 8 fev. 2021.
- SARKAR, Anirban *et al*. Android Application Development: A Brief Overview of Android Platforms and Evolution of Security Systems. *In*: INTERNATIONAL CONFERENCE ON I-SMAC (IOT IN SOCIAL, MOBILE, ANALYTICS AND CLOUD) (I-SMAC), 2019, Palladam, Índia. **Proceedings** [...]. [*S. l.*]: IEEE, 2019. p. 73-79. Disponível em: https://ieeexplore.ieee.org/document/9032440. Acesso em: 8 fev. 2021.

A segunda etapa consistiu na busca por mais títulos por meio do serviço de busca Google e pela ferramenta de busca de trabalhos acadêmicos Google Acadêmico. Novamente as palavras-chaves utilizadas foram "Android", "Operating System" e "Architecture". Nessa fase, mais alguns títulos foram selecionados, e vale citar que o ResearchGate foi a principal fonte.

Na terceira etapa, foram utilizados sites de venda, como o da Amazon, para procurar por livros sobre a estruturação do sistema Android. Ademais, também foi utilizada a base de dados SpringerLink para encontrar livros gratuitos sobre a arquitetura da plataforma. Mais uma vez as palavras-chaves consistiram em: "Android", "Operating System" e "Architecture". Para mais, nessa fase, alguns livros foram selecionados e, no caso dos livros pagos, foi empregada uma etapa adicional de obtenção, para que os livros pudessem ser selecionados.

A quarta e última etapa, foi realizada uma busca no Google pela documentação oficial do Android e da Java API da plataforma. Os portais encontrados foram o Android Developers e o Android Open Source Project. Os dois sites centralizam as novidades de cada versão, as explicações referentes a cada mecanismo das antigas e das novas abordagens para a arquitetura, e todas as Java APIs. Selecionou-se os dois para passar pela análise posterior.

Após a coleta dos dados, os materiais foram analisados, com o propósito de averiguar o casamento com as metas do trabalho em questão. De todos os materiais levantados, somente uma pequena parcela serviu como base teórica e, afim de comprovar a credibilidade e veracidade das informações, conforme indicado por Cristiane e Celia (2007), segue abaixo a referência dos livros utilizados junto com os sites, referenciados por meio da página inicial:

- ANDROID DEVELOPERS. **Documentação**: Sobre a Plataforma e SDK. Califórnia, 2020. Disponível em: https://developer.android.com/. Acesso em: 8 fev. 2021.
- ANDROID OPEN SOURCE PROJECT. **Documentação**: Código Aberto da Plataforma. Califórnia, 2020. Disponível em: https://source.android.com/. Acesso em: 8 fev. 2021.
- KRAJCI, Iggy; CUMMINGS, Darren. **Android on x86**: An Introduction to Optimizing for Intel Architecture. 1. ed. Berkeley, California: Apress, 2013. 380 p. Disponível em: https://link.springer.com/book/10.1007%2F978-1-4302-6131-5. Acesso em: 8 fev. 2021.
- MCHOES, Ann; FLYNN, Ida. Android Operating Systems. *In*: MCHOES, Ann; FLYNN, Ida. **Understanding Operating Systems**. 7. ed. Estados Unidos: Cengage Learning, 2013. cap. 16, p. 497-518.
- TANENBAUM, Andrew; BOS, Herbert. Estudo de caso 1: Unix, Linux e Android: Android. *In*: TANENBAUM, Andrew; BOS, Herbert. **Sistemas Operacionais Modernos**. 4. ed. São Paulo: Pearson Education do Brasil, 2016. cap. 10.8, p. 555-589.

Com os documentos e livros acima, efetivou-se a criação de diferentes tópicos a respeito do tema: a história e surgimento do Android; o histórico de versões e as novidades de cada versão, até a verão mais atual, considerando o ano vigente no momento de realização deste trabalho; e a descrição detalhada da arquitetura do sistema, que é foco de todo o estudo.

4 Considerações Finais

O objetivo deste trabalho foi realizar a separação de cada camada da arquitetura do sistema operacional Android e trazer uma explicação do tanto do funcionamento individual dos principais componentes da mesma, bem como o funcionamento em conjunto.

A importância e relevância deste tema consiste em que o Android tem crescido exponencialmente no mundo todo atualmente. Por conta disso, compreender como funciona esse sistema operacional serve tanto aos desenvolvedores novos de aplicações do Android, quanto para os leigos a respeito da plataforma. Portanto, é possibilitado uma maior difusão das informações a respeito da arquitetura desse sistema.

O primeiro passo do trabalho foi encontrar fontes de pesquisa que garantissem a confiabilidade e que fossem relevantes no meio acadêmico. Em um primeiro momento, dois artigos e um livro foram selecionados para compor a base de pesquisa desta monografia, sendo eles: "Welcome to Binder: A kernel level attack model for the Binder in Android operating system", dos autores Maijd Salehi, Farid Daryabar e Mohammad Hesam Tadayson; "Android Application Development: A Brief Overview of Android Platforms and Evolution of Security Systems", de diversos autores e; "Sistemas Operacionais Modernos", do autores renomados Andrew Tanenbaum e Herbert Bos. Diante disso, percebeu-se que a maioria das bases cientificas famosas de computação, como IEEE e ACM, não obtiveram resultados relevantes para as palavras-chave "Android" e "Architecture", ocasionando na baixa oferta de artigos científicos dessas bases para a composição do trabalho.

A partir disso, optou-se pela pesquisa numa base de dados geral, no caso o Google Acadêmico, e que, felizmente, resultou em duas referências relevantes e importantes para a composição deste trabalho: "Android on x86: Na Introduction to Optimizing for Intel Architecture", dos autores Iggy Krajci e Darren Cummings e; "Understanding Operating Systems", das autoras Ann Mchoes e Ida Flynn. Além disso, a outra fonte fundamental para a composição do trabalho se deu pelos próprios desenvolvedores do sistema operacional Android: a documentação oficial, presente tanto no site "Android Developers" como no "Android Source".

Diante disso, as informações a respeito da arquitetura foram coletadas e analisadas, a fim de se extrair o máximo de informações relevantes sobre as camadas e como elas funcionavam entre si. A documentação oficial e o livro de Tanenbaum foram as principais fontes utilizadas, pois detinham a maior quantidade de detalhes do funcionamento das estruturas. Em meio a isso, o trabalho foi produzido, abordando as seis diferentes camadas da arquitetura: Kernel Linux; Camada de Abstração de Hardware; Android Runtime; Bibliotecas Nativas C/C++; Framework da Java API e; Aplicações.

Em síntese, compreender o funcionamento interno do sistema operacional Android

como um todo, bem como explicitar as funções específicas de cada componente da arquitetura, possibilita uma maior difusão das informações tanto para aqueles que são leigos neste assunto quanto para os novos desenvolvedores de aplicações. Portanto, os objetivos foram atingidos, e espera-se que esse trabalho sirva como base na construção de mais artigos científicos a respeito da arquitetura desse importante sistema operacional.

Referências

BOOTLIN. *Android System Development*. 2019. Disponível em: https://bootlin.com/doc/legacy/android//android-slides.pdf. Acesso em: 11 fev. 2021.

BRAHLER, S. Analysis of the android architecture. *Karlsruhe institute for technology*, v. 7, n. 8, 2010. Disponível em: https://www.it.iitb.ac.in/frg/wiki/images/2/20/2010_braehler-stefan_android_architecture.pdf. Acesso em: 11 fev. 2021.

CRISTIANE, T.; CELIA, R. Procedimentos metodológicos na construção do conhecimento científico: a pesquisa bibliográfica. *Revista Katálysis*, scielo, v. 10, p. 37 – 45, 00 2007. ISSN 1414-4980. Disponível em: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1414-49802007000300004&nrm=iso.

DEVELOPERS, A. *Documentação: Sobre a Plataforma e SDK (2020)*. 2020. Disponível em: https://developer.android.com/. Acesso em: 11 fev. 2021.

DEVELOPERS, A. *Documentação: Sobre a Plataforma e SDK (2021)*. 2020. Disponível em: https://developer.android.com/. Acesso em: 11 fev. 2021.

DRAKE, J. J.; LANIER, Z.; MULLINER, C.; FORA, P. O.; RIDLEY, S. A.; WICHERSKI, G. *Android hacker's handbook*. [S.l.]: John Wiley & Sons, 2014.

GUNASEKERA, S. *Android apps security*. [S.l.]: Springer, 2012.

KRAJCI, I.; CUMMINGS, D. *Android on x86: An Introduction to Optimizing for Intel® Architecture*. [S.l.]: Springer Nature, 2013.

MCHOES, A.; FLYNN, I. M. *Understanding Operating Systems*. [S.l.]: Cengage Learning, 2013. v. 7.

PAGE, L. *Update from the CEO*. 2013. Disponível em: https://blog.google/products/android/update-from-ceo/?m=1. Acesso em: 11 fev. 2021.

PIZZANI, L.; SILVA, R. C. da; BELLO, S. F.; HAYASHI, M. C. P. I. A arte da pesquisa bibliográfica na busca do conhecimento. *RDBCI: Revista Digital de Biblioteconomia e Ciência da Informação*, v. 10, n. 2, p. 53–66, 2012.

TANENBAUM, A. S.; BOS, H. *Sistemas Operacionais Modernos*. [S.l.]: Pearson, 2016. v. 4.