UNIVERSIDADE FEDERAL FLUMINENSE – UFF
INSTITUTO DE CIÊNCIA E TECNOLOGIA – ICT
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Caio Borghi Falco

ESTUDO DO SISTEMA OPERACIONAL ANDROID PARA DISPOSITIVOS MÓVEIS

Rio das Ostras – RJ

2017

Caio Borghi Falco

ESTUDO DO SISTEMA OPERACIONAL ANDROID PARA DISPOSITIVOS MÓVEIS

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense como parte dos requisitos para obtenção do Grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Marcos Ribeiro Quinet de Andrade

Rio das Ostras – RJ

2017

Caio Borghi Falco

ESTUDO DO SISTEMA OPERACIONAL ANDROID PARA DISPOSITIVOS MÓVEIS

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense como parte dos requisitos para obtenção do Grau de Bacharel em Ciência da Computação.

Aprovado em dezenove de dezembro de 2017.

BANCA EXAMINADORA

_____

Prof. Dr. Marcos Ribeiro Quinet de Andrade – Orientador UFF

_____

Prof. Dr. Luciano Bertini – Avaliador UFF

_____

Prof. Dr. Alessandro Copetti – Avaliador UFF

Rio das Ostras – RJ

2017

Aos meus pais.

AGRADECIMENTOS

Aos meus pais, pelo apoio incondicional e completo suporte durante este ciclo.

À minha namorada Larissa Ferreira e aos amigos da “Mulekadinha” que tornaram minha trajetória menos árdua.

Ao professor Marcos Quinet pela orientação.

RESUMO

Os dispositivos de comunicação móveis passaram por um processo de maciça popularização, particularmente acentuado na última década. Como consequência, o surgimento de sistemas operacionais para esses aparelhos também experimentou um ciclo de desenvolvimento bastante ágil, com atualizações constantes para suportar novos recursos. Este trabalho propõe-se a apresentar uma análise dos sistemas operacionais para dispositivos móveis, focando principalmente no sistema Android, que mostrou-se o mais interessante de ser estudado por uma série de razões, entre elas, por ser *open source*, por ser atualmente o sistema mais adotado globalmente, por estar em constante aprimoramento e ter integração com outros dispositivos além de um *smartphone*. Este trabalho propõe-se a apresentar uma análise abrangente da estrutura e funcionamento do Android, além de seus princípios básicos de gerenciamento dos recursos de *hardware* e a interface para suportar as aplicações do usuário.

Palavras-chaves:
Sistemas operacionais móveis, sistemas operacionais, Android, Linux, iOS, gerenciamento de recursos, escalonamento de processos.

ABSTRACT

As the mobile communication devices underwent a process of massive popularization, particularly accentuated in the last decade, the development of operating systems for these devices have also experienced a fairly rapid cycle, with constant updates to support new features. This term paper proposes to present an analysis of the operating systems for mobile devices, focusing mainly on the Android system, which for a number of factors proved to be the most interesting, among them being open source, being the most used system in global level, being constantly improved and having integration with other devices besides a smartphone. This term paper is going to present an in-depth analysis of the structure and operation of Android, as well as its basic principles of hardware resource management and the interface to support user applications.

Keywords:
Mobile operating systems, operating systems, Android, Linux, iOS, resource management, task scheduling.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

PDA: Personal Digital Assistant
PIM: Personal Information Manager
GPS: Global Positioning System
SDK: Software Development Kit
IDE: Integrated Development Environment
3G: 3rd Generation
API: Application Programming Interface
APK: Android Package
URI: Uniform Resource Identifier
ART: Android Runtime
UID: User Identification
JIT: Just in Time
AOT: Ahead of Time
ELF: Executable and Linkable Format
UCP: Unidade Central de Processamento
I/O: Input/Output
CPU: Central Processing Unit
FIFO: First In First Out
RR: Round Robin
FAT: File Allocation Table
ExFAT: Extended File Allocation Table
NTFS: New Technology File System
EXT2: Extended File System

VFS: Virtual File System

F2FS: Flash-Friendly File System

JFF2: Journal Flash File System

LRU: Least Recently Used

HAL: Hardware Abstraction Layer

SSL: Secure Sockets Layer

JVM: Java Virtual Machine

SUMÁRIO

1. INTRODUÇÃO

Estudos sobre sistemas operacionais e todos os conceitos e princípios que estes abrangem não é algo recente. Já são várias décadas de pesquisas que embasaram o desenvolvimento de inúmeras técnicas sofisticadas para a utilização otimizada da unidade central de processamento, controle de operações de entrada e saída, gerenciamento de memória, escalonamento de processos, além do constante aprimoramento da segurança e de muitos outros tópicos que o sistema operacional deve se encarregar. Entretanto, quando um novo contexto emerge de forma avassaladora, muitas questões precisam ser revistas. O Android e os dispositivos móveis representam este contexto. Seja por meio de um *smartphone*, *tablet* ou qualquer outro dispositivo móvel semelhante, sua utilização apresenta importantes diferenças quando comparada a utilização de sistemas computacionais tradicionais (como computadores de mesa, por exemplo). Questões como gerenciamento de energia, experiência do usuário, abertura (quanto do código deve ser disponibilizado ao público) e tempo de resposta passaram a ser cruciais de modo que novas técnicas para lidar com estas questões precisaram ser desenvolvidas em busca de uma maior satisfação do usuário. Em relação a estrutura do trabalho, esta foi dividida em cinco capítulos e caminha de uma abordagem superficial para uma cada vez mais profunda; o primeiro capítulo apresenta introdução e objetivos do trabalho, o segundo descreve a evolução dos sistemas operacionais móveis abordando os principais ao longo desta trajetória. O terceiro faz uma análise acerca da distribuição de usuários dos sistemas operacionais móveis e avalia tópicos relevantes como desenvolvimento de aplicações e assistentes pessoais virtuais. O quarto e mais importante capítulo analisa os principais conceitos que englobam o complexo sistema Android. A monografia é concluída com as considerações finais presentes no quinto capítulo.

1.1. OBJETIVOS

Este trabalho propõe-se a realizar um abrangente estudo sobre o sistema operacional Android, enfatizando suas principais diferenças em relação a sistemas operacionais para sistemas computacionais tradicionais, bem como sua grande

adaptabilidade ao estar presente nos mais diversos tipos de dispositivos móveis (sejam *smartphones* de diferentes fabricantes ou em outros “ecossistemas” como *smartwatchs*, *videogames,* telefones fixos e até espelhos) além de seu grande sucesso, cujo ápice aconteceu quando o Android superou o Windows como o sistema operacional mais popular do mundo em abril de 2017 (G1, 2017).

2. HISTÓRICO E EVOLUÇÃO DOS SISTEMAS OPERACIONAIS PARA DISPOSITIVOS MÓVEIS

Este capítulo propõe-se a apresentar os principais sistemas operacionais móveis em ordem cronológica. Cada um deles é explorado superficialmente de forma que apenas as suas principais características sejam compreendidas, bem como a forma como os sistemas operacionais móveis foram se tornando cada vez mais complexos ao longo do tempo.

2.1. ROM-DOS

O primeiro dispositivo considerado um *smartphone* foi produzido em 1994 pela IBM e chamava-se IBM Simon. O dispositivo apresentava tela sensível ao toque, e-mail e recursos de PDA (*Personal Digital Assistant*). O seu sistema operacional era o ROM-DOS (também chamado de Datalight DOS), que foi introduzido em 1989 como um sistema operacional compatível com o MS-DOS projetado para sistemas embarcados.

2.2. PALM OS

Em 1996, o PDA Palm Pilot 1000 é lançado com o Palm OS, que é considerado o primeiro sistema operacional móvel. O Palm OS oferecia suporte a tela sensível ao toque e incluía um conjunto de aplicações básicas para gerenciamento de informações pessoais. Sua primeira versão oferecia serviços como "*address*", "*date book*", "*memo pad*" e "*to do list*" e uma calculadora. Inúmeras atualizações do Palm OS foram lançadas, inclusive com suporte para *smartphones*.

Foi descontinuado após sua última atualização, ocorrida em outubro de 2007. A tela principal, ilustrada pela figura 1, apesar de atualmente não causar grande impacto, foi uma novidade para a época e desde então passou a ser uma espécie de padrão para os dispositivos seguintes (PALM OS, 2010).

Figura 1. *Tela principal do Palm OS.*
Fonte: (MCLOUGHLIN, 2009).

2.3. WINDOWS CE

Em novembro de 1996 a Microsoft lançou o Windows CE (que posteriormente seria renomeado para Windows Embedded Compact). Sua primeira versão executava em computadores de mão chamados *handheld PCs,* representados pela figura 2, e oferecia aplicações como o Pocket Word e o Pocket Excel, além de recursos de PIM (*Personal Information Manager*).

Muitas plataformas basearam-se no sistema operacional do Windows CE, incluindo o AutoPC, o Pocket PC 2000 da Microsoft, o Pocket PC 2002, o Windows Mobile 2003, o Windows Mobile 5 e muitos dispositivos industriais e sistemas embarcados.

A Microsoft licencia o Windows Embedded Compact aos fabricantes de equipamentos, que podem modificar e criar suas próprias interfaces com o Windows CE fornecendo os recursos para tal.

O Windows Embedded Compact é otimizado para dispositivos com memória mínima; Um *kernel* do Windows CE pode ser executado ocupando apenas um megabyte de memória do dispositivo.

Hoje em sua sétima versão, o Windows Embedded Compact 7 é voltado para ferramentas específicas como câmeras digitais, sistemas de GPS, controladores industriais, sistemas de entretenimento automotivo, entre outros.

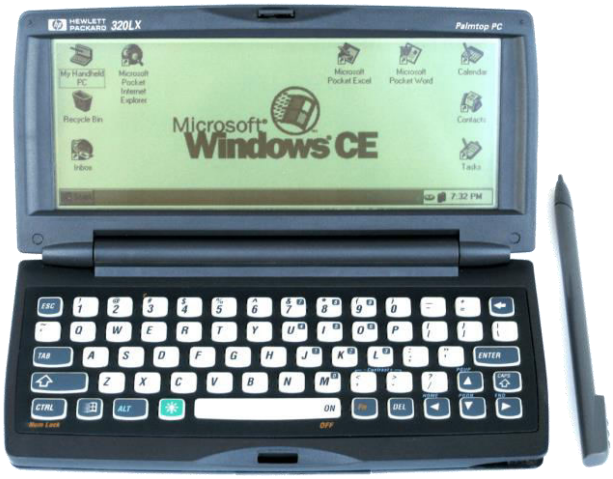

Figura 2. *Handheld PC com Windows CE.*
Fonte: (ALGAZE, 2015).

2.4. SERIES 40

A Séries 40, ou simplesmente S40, foi lançada em 1999 para os telefones celulares Nokia 7110, ilustrado pela figura 3. O S40 funcionava em dispositivos mais simples da Nokia (os *smartphones* utilizavam Symbian até 2012 e Windows Phone desde então) e oferecia recursos básicos como mensagens, e-mail, aplicações de câmera, gravação de vídeo, rádio, calendário, entre outros.

Observa-se que o S40, não é um sistema operacional em si, mas uma plataforma de software e interface de usuário da Nokia, com uma estrutura muito simplificada para ser considerado como um sistema operacional completo.

Foi usado em grande escala mundialmente, chegando a mais de 1.5 bilhão de dispositivos vendidos, segundo dados da Nokia em 2012 (WEBER, 2012).

Figura 3. *Nokia 7110.*
Fonte: (CHINADAILY, 2014).

2.5. SYMBIAN

A origem do Symbian remete a empresa inglesa Psion, que em 1984 passou a produzir computadores de mão (*handheld PCs*) com um sistema operacional próprio chamado EPOC. Em 1998 formou-se a Symbian Ltd, empresa desenvolvedora do Symbian OS que tinha como membros a Psion, Ericsson, Motorola e Nokia, cujo principal produto não eram os computadores de mão (que não haviam conquistado o apelo esperado e foram descontinuados em 2001), mas o sistema operacional EPOC, que deu origem ao Symbian.

Em 2000, o Symbian tornou-se o primeiro sistema operacional móvel moderno em um *smartphone* com o lançamento do Ericcson R380. Originalmente desenvolvido como um sistema *closed-source* para PDAs em 1998, foi usado por muitas das maiores marcas de celulares como Samsung, Motorola, Sony Ericsson e, sobretudo, pela Nokia. Como um pioneiro que estabeleceu uma interface para a indústria de *smartphones*, foi o sistema operacional mais popular em escala mundial até o fim de 2010, quando o Android assumiu a liderança. O Symbian (de 2001) foi essencialmente, um *shell system* que exigia uma interface de usuário para formar um sistema operacional completo.

A partir de 2010, Symbian deixa de ser uma versão específica de C++ com a IDE Carbide.c++ como ambiente de desenvolvimento de aplicações nativo e passa a usar C++ padrão com Qt como o SDK principal. Outras linguagens que poderiam ser

utilizadas para desenvolvimento de aplicações são Python, Java ME, Ruby, .NET, entre outras.

Em fevereiro de 2011, a Nokia, era a única companhia que ainda dava suporte ao Symbian fora do Japão, anunciou que iria a partir de então usar o Windows Phone 7 como sua principal plataforma enquanto o Symbian iria gradualmente sendo desativado. Atualmente, o sistema encontra-se descontinuado (MORIMOTO, 2010).

2.6. POCKET PC 2000

O Pocket PC 2000 lançado no mesmo ano (renomeado em 2003 como "Windows Mobile") foi baseado no *kernel* do Windows CE. Em 2007, era o sistema operacional para *smartphones* mais popular nos EUA, mas essa popularidade desapareceu nos anos seguintes. Em fevereiro de 2010, enfrentando uma forte concorrência de sistemas operacionais rivais, incluindo o iOS e Android, a Microsoft anunciou que o Windows Phone iria substituir o Windows Mobile, causando sua descontinuidade.

A primeira versão apresentava vários aplicativos embutidos, entre eles o Microsoft Reader, Windows Media Player além de outros, como o Pocket Word e o Pocket Excel, que eram versões dos softwares do pacote Microsoft Office.

2.7. IOS

Lançado em 2007 com o primeiro iPhone, o iOS teve sua primeira versão apresentada ao mundo. A esta altura, o sistema operacional móvel ainda levava o nome de iPhone OS; apenas em sua quarta versão (lançada em 2010) teve seu nome alterado para iOS. Ainda em 2007, um mês após seu lançamento, a Apple anunciou o novo iPod Touch, que também fazia uso do iPhone OS.

O iPhone OS, foi baseado no sistema da Apple para computadores (Mac OS) e não suportava conexões 3G, não apresentava multitarefa, mensagens multimídia e sequer a possibilidade de copiar (ou recortar) e colar um texto. Também não era possível personalizar sua tela inicial. Além disso, só era possível utilizar os

aplicativos que vinham instalados de fábrica, não havia a possibilidade de baixar aplicativos desenvolvidos por terceiros.

A ausência de recursos básicos pouco importava naquele momento, principalmente porque a Apple não vinha com uma proposta de competir com recursos e ferramentas, mas de apresentar um *smartphone* com o melhor funcionamento e experiência de usuário do mercado, e nestes quesitos a empresa de fato obteve sucesso.

Um dos grandes diferenciais em sua primeira versão foi o navegador Safari. Nesta época, os sistemas operacionais móveis concorrentes não possuíam um navegador tão sofisticado como o Safari, que permitia recursos de zoom e rolagem que eram únicos para a época. Após duas atualizações, passou a ser possível comprar músicas via iTunes.

A segunda versão do iPhone OS lançada em julho de 2008 trouxe novidades. Foi lançada a App Store, que possibilitava aos usuários baixarem aplicativos desenvolvidos por terceiros. Os desenvolvedores também se beneficiaram disso, já que a Apple passou a fornecer um kit de desenvolvimento para o sistema operacional. Com isso, jogos em 3D se popularizaram e os aplicativos do iPhone estavam se tornando mais bonitos, funcionais e sofisticados do que os vistos em qualquer outra plataforma da época.

Em sua terceira versão, lançada em 2009, as principais novidades foram a inclusão de gravação de vídeo, melhora da busca de conteúdos dentro dos seus dispositivos e a possibilidade de selecionar textos para copiar, recortar e colar. O foco para a quarta versão foi fazer do sistema, agora rebatizado como iOS, um sistema operacional multitarefa (suporte a aplicativos executando em paralelo de forma consistente). Certamente a principal novidade do iOS 5 foi a Siri (ainda em sua versão beta), uma assistente virtual capaz de realizar diversas tarefas, como realizar chamadas telefônicas, buscas na internet, entre outras facilidades.

Em 2010, foi anunciado que o iOS passaria a ser embarcado também no iPad, o *tablet* da Apple. O sistema operacional passou por adaptações para ser utilizado com outros dispositivos da Apple, como o Apple Watch e Apple TV. Apenas até a sua terceira geração a Apple TV fez uso do iOS, após isso recebeu seu próprio sistema operacional, o tvOS.

Em 2012 a disputa entre a Apple/iOS e Google/Android pelo mercado de dispositivos móveis era ferrenha, e ao promover a saída do Google Maps no iOS 6 isso ficou claro. No entanto, a Apple não ficaria sem um aplicativo de mapa e essa versão do iOS apresentou como novidade o aplicativo ‘Mapa’, de propriedade da Apple. Ainda nessa versão a Siri foi consideravelmente aprimorada, com a inclusão de muitos novos idiomas e informações sobre filmes, ligas esportivas, programas de TV, além da possibilidade de executar aplicativos e postar mensagens nas redes sociais.

O destaque do iOS 7 foi a mudança mais significativa de interface gráfica entre uma versão e sua anterior desde sua versão 1.0. Todos os ícones de aplicativos nativos ganharam novo design e layouts internos (MEYER, 2016) (ZAP, 2015).

Em 2014 foi lançado o iOS 8, que como destaque apresentou a melhoria de alguns aplicativos como o ‘Mensagens’ e o ‘Fotos’, além do surgimento do aplicativo ‘Saúde’, que permite um acompanhamento da saúde do usuário através de informações médicas, níveis biológicos, estatísticas de atividades físicas, do sono, etc. A versão 8.3 do sistema operacional passou a oferecer compatibilidade com o Apple Watch. A sua nona versão, lançada em 2015, aprimorou ainda mais a assistente virtual Siri; agora ela pode fazer sugestões antes mesmo do usuário solicitar e consegue fazer buscas mais precisas no iOS como datas, fotos de contatos, etc. Também passou a ser possível ativar o modo "Pouca Energia", que faz com que o dispositivo consuma menos energia estendendo assim a duração de sua bateria (PESTRE, 2015).

O iOS 10 foi lançado em setembro de 2016, e novamente a Siri fez parte das principais novidades, podendo ser utilizada por desenvolvedores terceiros. Ela também foi implementada no teclado do iOS o que permite ao usuário receber sugestões como compartilhar localização, informações de contatos e dados do calendário, tudo baseado com o que acontece na sua conversa. Alguns aplicativos nativos tiveram seu design remodelado como o Apple Music, Apple Maps e Apple News. Outra novidade foi o aplicativo Home. Através dele tornou-se possível controlar todos os acessórios inteligentes de sua casa. É possível, por exemplo, apagar as luzes, ver quem está tocando a campainha ou controlar remotamente a Apple TV.

O iOS 11 está com lançamento previsto para o terceiro trimestre de 2017. Algumas das principais novidades divulgadas são a função "*Do Not Disturb While Driving*" que detecta quando você está dirigindo e evita notificações ao longo do percurso (é possível definir contatos prioritários para que suas mensagens ou ligações não sejam ignoradas) e também mapas *indoor*. Alguns locais como shoppings e aeroportos terão mapas no iOS 11 de modo a auxiliar o usuário, caso ele necessite de informações sobre estes locais (MULLER, 2016) (CIRIACO, 2017).

2.8. ANDROID

A Android Inc, fundada em 2003, era uma empresa de software cujo objetivo inicial era desenvolver um software para tornar os dispositivos móveis mais inteligentes. A empresa inicialmente tinha seu foco em câmeras digitais, mas logo teve sua atenção voltada para os *smartphones* por conta de seu grande - e maior - potencial de mercado. Com a mudança de rumo, a meta passou a ser lidar com a dificuldade que existia na época em desenvolver para dispositivos móveis, trazendo a eles uma plataforma aberta construída sobre o Linux que pudesse ser amplamente usada.

Três linguagens de programação eram os alvos de modo a dar suporte a um rico ambiente de desenvolvimento de aplicativos: Java, JavaScript e C++.

No entanto, no início de 2006, o Google (que adquirira a Android Inc. em julho de 2005) mudou consideravelmente o plano: a plataforma focaria exclusivamente na linguagem Java para o desenvolvimento de aplicativos. Essa decisão baseou-se na ideia de que ao construir apenas uma API (*Application Programming Interface*) de desenvolvimento (ao invés de três) reduziria significativamente o tempo e o esforço necessário, além de possibilitar um avanço na qualidade.

A primeira disponibilização pública do Android foi uma pré-estreia do SDK, lançado em novembro de 2007. Ele consistia em um emulador de dispositivo de *hardware* executando um sistema de dispositivo Android completo em termos de aplicativos de imagem e núcleo, documentação de API e um ambiente de desenvolvimento.

As atividades de desenvolvimento seguiram por meio de duas linhas: terminar e estabilizar a implementação para lançar o dispositivo HTC-Dream e analisar o *feedback* a respeito do SDK para aprimorar ainda mais as APIs. Em agosto de 2008, foi lançada a versão 0.9 do SDK que continha as APIs praticamente finais. Ainda no mesmo mês, com o software estável e pronto, o projeto foi enviado para fábrica e os dispositivos começaram a ser produzidos. Em setembro, o HTC-Dream também conhecido como T-Mobile G1 é lançado com o Android 1.0.

O desenvolvimento continuou em ritmo acelerado após o lançamento do Android 1.0. Durante os cinco anos seguintes, cerca de 15 importantes atualizações foram lançadas, aprimorando as características da versão 1.0, bem como adicionando novidades (TANENBAUM e BOS, 2016).

A versão 1.5 do Android - a primeira a receber apelido de sobremesa, o que virou padrão desde então - chamada de *Cupcake*, teve a inclusão de *widgets*, gravação e reprodução de vídeos em formato MPEG-4 e 3GP e melhorias no teclado, que passou a funcionar tanto na horizontal quanto na vertical. A versão 2.0 chamada *Eclair* ("bomba de chocolate"), foi lançada em 2009 e foi a primeira atualização radical do sistema operacional móvel. Trouxe uma nova interface, velocidade otimizada e suporte ao HTML5 no navegador. As versões de 3.0 a 3.2 chamadas de *Honeycomb* ("favo de mel") foram destinadas a *tablets*. A maioria das Smart TVs com o sistema Google TV utilizava uma versão modificada do *Honeycomb* 3.2 (ALVES, 2015) (KLEINA, 2017) (SANTINO, 2013).

A versão 6.0 (*Marshmallow*) chegou em 2015 trazendo algumas novidades, destacando os modos "Não Perturbe" e "Doze", que otimiza a bateria do dispositivo quando está em "*stand-by*". A versão 7.0 (*Nougat*) foi lançada em agosto de 2016. Através do "*Daydream*" que é uma plataforma de realidade virtual nativa do Android, é possível abrir aplicativos e jogos através de um óculos de realidade virtual. Outra novidade é o Modo de Economia de Dados, que toma medidas para consumir menos o pacote de dados, através da reprodução de vídeos em menor resolução, restrição no envio e recebimento de mensagens em segundo plano e carregamento de fotos no navegador apenas se forem tocadas.

Em 21 de agosto de 2017, foram divulgadas melhorias e novidades para o Android 8.0 (Oreo). O Google garante que o tempo de inicialização dos dispositivos será consideravelmente reduzido. Através do Picture-in-Picture será possível

minimizar um aplicativo para um dos cantos do display enquanto utiliza outro por trás normalmente (HIGA, 2016) (MULLER, 2017).

Por ser de código aberto e altamente customizável, o Android se faz presente em muitos dispositivos além de smartphones e *tablets*. Sua abrangência é tão grande que inclui itens como Smart TVs, câmeras, relógios inteligentes, tocadores de CD e DVD de carros e até mesmo espelhos (HOLLISTER, 2012) (SILVA, 2012).

Em janeiro de 2011 foi noticiado que o número de vendas de dispositivos Android no terceiro trimestre de 2010 superou o do Symbian e se tornou a plataforma mais popular no mercado de smartphones. Desde então ele não perdeu mais a liderança e foi se consolidando cada vez mais (REUTERS, 2011).

2.9. WINDOWS PHONE

O Windows Phone é o sistema operacional para *smartphones* desenvolvido pela Microsoft. Lançado no final de 2010, o nome oficial do sistema operacional era Windows Phone 7 Series, mas após uma repercussão negativa devido ao nome ser considerado muito extenso, tornou-se apenas Windows Phone.

No primeiro semestre de 2011, a Microsoft firmou parcerias com algumas empresas como Nokia, Acer, Fujitsu e ZTE. Todas empresas planejavam lançar o primeiro dispositivo quando acontecesse a primeira grande atualização da plataforma, denominada Windows Phone 7.5 (Mango).

Com a chegada do Windows Phone 8 (Apollo) em outubro de 2012, uma grande mudança ficou evidente. Dispositivos com o Windows Phone 7 não poderiam ser atualizados para o Windows Phone 8 e novas aplicações desenvolvidas especificamente para o Windows Phone 8 não eram compatíveis com os dispositivos Windows Phone 7.x.

Quando menos de 1% dos smartphones vendidos no mundo no primeiro trimestre de 2016 eram Windows, ficou claro que mesmo com a impactante atualização do Windows Phone 8.1, a plataforma não foi tão bem recebida como esperado.

A Microsoft então decide voltar suas atenções para o Windows 10 Mobile, que é uma versão móvel do seu homônimo para computadores. O Windows 10

Mobile surgiu com o objetivo de ter uma maior integração com a sua versão para computadores pessoais. Com o advento dessa nova versão, a Microsoft já divulgou oficialmente que o suporte oficial ao Windows Phone 8.1 está encerrado (SANTINO, 2016) (DAVIES, 2011) (RUBINO, 2012) (G1, 2017).

2.10. ABERTURA

A questão da abertura ("*openness*") dos sistemas operacionais móveis mostrou-se importante para o desenvolvimento dessa área da computação. Há alguns anos, a maioria dos dispositivos móveis possuía basicamente as funções de realizar ligações e enviar mensagens de texto. Para os usuários, apenas as aplicações que já acompanhavam o dispositivo quando este saía da fábrica eram as que ele poderia utilizar. Os desenvolvedores de aplicações não tinham acesso a nenhuma parte do código fonte do sistema operacional sem um contrato com seu proprietário. Os desenvolvedores de aplicações precisavam trabalhar próximos aos desenvolvedores do sistema operacional, já que apenas estes sabiam como desenvolver aplicações compatíveis com o sistema. Com a transição de telefones móveis para *smartphones*, as pessoas desejavam que seu dispositivo fosse capaz de fazer mais, como navegar na internet, tocar músicas, vídeos, etc. Com o intuito de encorajar mais desenvolvedores a satisfazerem o anseio dos usuários, os criadores de sistemas operacionais móveis simplesmente forneciam conjuntos de APIs e ferramentas relacionadas como SDKs para que as pessoas pudessem desenvolver qualquer tipo de aplicação para os dispositivos móveis. Com tanta abertura, os desenvolvedores conseguiram a liberdade de desenvolver aplicações para sistemas operacionais móveis, o que tornou possível para os consumidores comprar e instalar mais aplicações do que apenas aquelas que já acompanhavam o aparelho. Em função de tamanho sucesso com os desenvolvedores e consumidores, tornou-se praticamente “obrigatório” para a maioria dos sistemas operacionais móveis fornecer um conjunto de APIs e um SDK. Nos anos recentes, mais e mais sistemas operacionais móveis tem sido desenvolvidos com todo seu código fonte disponível para o público, além do acesso às APIs e SDKs. Qualquer pessoa tem a possibilidade de ver todo o código fonte, contribuir e personalizar o sistema operacional. (LI, WANG, *et al.*, 2012).

3. COMPARAÇÕES GERAIS ENTRE OS SISTEMAS OPERACIONAIS MÓVEIS

Este capítulo aborda tópicos importantes associados aos sistemas operacionais móveis como distribuição de usuários, experiência de usuário, desenvolvimento de aplicações, assistentes pessoais virtuais e integração com dispositivos diferentes de um *smartphone*. Cada seção visa deixar claras as principais diferenças entre os grandes sistemas operacionais móveis de hoje.

3.1. DISTRIBUIÇÃO DE USUÁRIOS

As informações apresentadas nesta seção foram retiradas da ferramenta de tráfego StatCounter, que monitora cerca de 2 milhões de páginas *web* (STATCOUNTER, 2017). Baseando-se nos acessos realizados a estas páginas, a ferramenta calcula a porcentagem de acesso de cada sistema operacional.

Durante o ano de 2009, a fatia de mercado ocupada pelos sistemas operacionais móveis oscilou consideravelmente. É certo que o Symbian iniciou (cerca de 39%) e terminou (cerca de 35%) o ano com a maior parte do mercado, no entanto, em julho chegou a registrar aproximadamente 24% do mercado, momento em que o iOS liderava, com 34% do total. Neste momento, o mercado mostrava-se muito particionado, com vários outros sistemas operacionais móveis apresentando mais de 1% do mercado, entre eles, Windows (1,2%), Playstation (1,6%), Android (4%), Sony Ericsson (7,1%) e BlackBerry OS (9,7%), além do iOS com 32%, sendo o vice-líder. O mercado manteve-se bem dividido em 2010, com destaque para o crescimento de Android e do BlackBerry OS, que fecharam o ano com 13,6% e 18% do mercado, respectivamente, e a queda do iOS, que no mesmo momento apresentava 23,6%.

Em 2011, pela primeira vez foi possível ver o início de uma polarização. Ao término do ano, 80% dos smartphones vendidos no mundo utilizavam o sistema operacional Symbian, iOS ou Android. Em 2011, foi notável o crescimento do Android, que não só fechou o ano na terceira colocação com 21,8% do mercado, como chegou a alcançar a vice-liderança por cerca de 45 dias quando ultrapassou o iOS entre julho e o final de agosto.

O ano de 2012 foi marcante para o mercado de smartphones, pois observou-se a acentuada queda de popularidade do Symbian. Analisando a figura 4, verifica-se que ele começou o ano como o mais popular, com 31,9% do mercado, e terminou na quarta posição, com apenas 10,7%. O Android herdou a liderança do Symbian, sendo seguido pelo iOS. A terceira colocação ficou com o surpreendente Series 40, que teve ascensão meteórica.

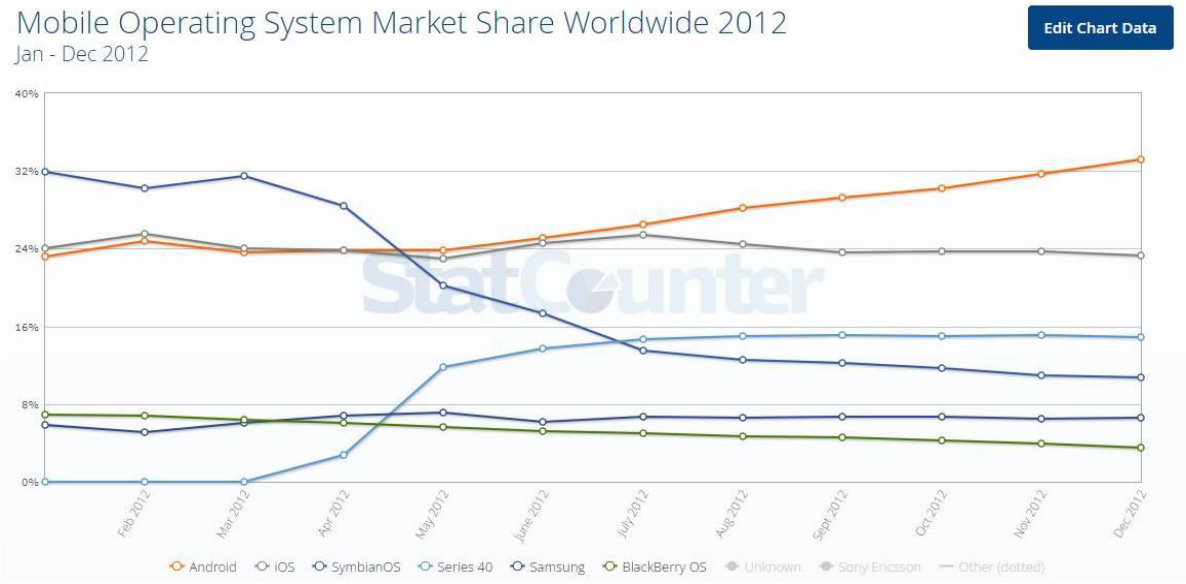

Figura 4. *Distribuição de uso de sistemas operacionais móveis em 2012.*
Fonte: (STATCOUNTER, 2012).

Observando os dados apresentados pelas figuras 5, 6 e 7, é possível afirmar que os anos de 2013, 2014 e 2015 apresentaram um comportamento similar quanto a consolidação do Android como plataforma dominante do mercado mundial e a decadência tanto do S40 quanto do Symbian, estabelecendo uma nítida polarização do mercado.

O Android iniciou 2013 com 36,9% do mercado, e ao término de 2015 apresentava 65,9%. O iOS apresentou 25,9% e 19,2% nos mesmos períodos, e o Windows Phone continuou com uma participação discreta durante o período, com as porcentagens de 1,1% e 2,3% nos períodos indicados.

Figura 5. *Distribuição de uso de sistemas operacionais móveis em 2013.*

Fonte: (STATCOUNTER, 2013).

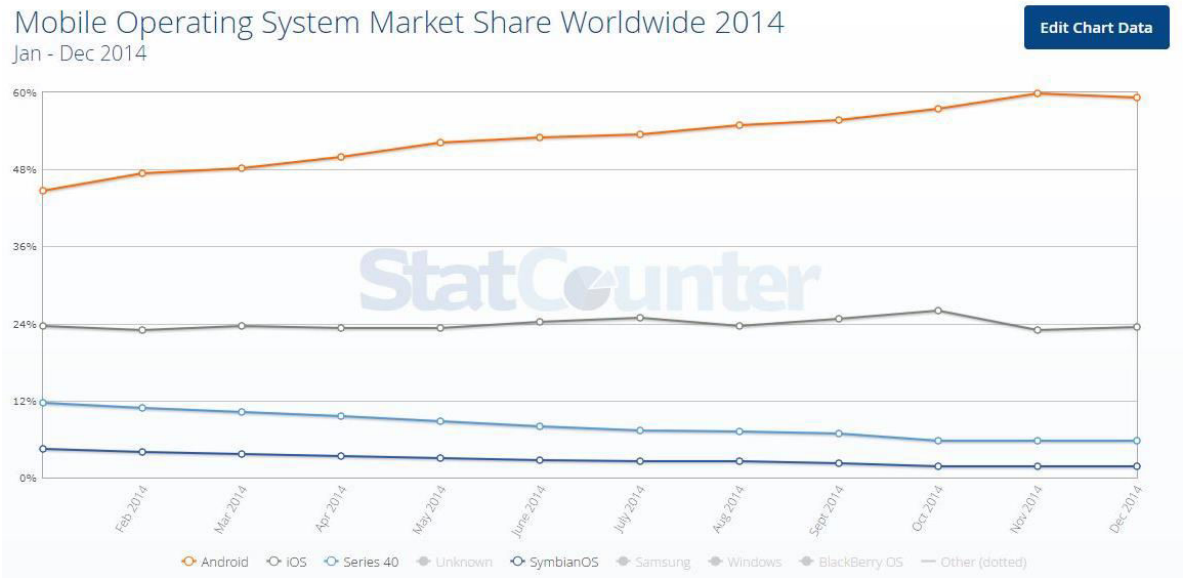

Figura 6. *Distribuição de uso de sistemas operacionais móveis em 2014.*

Fonte: (STATCOUNTER, 2014).

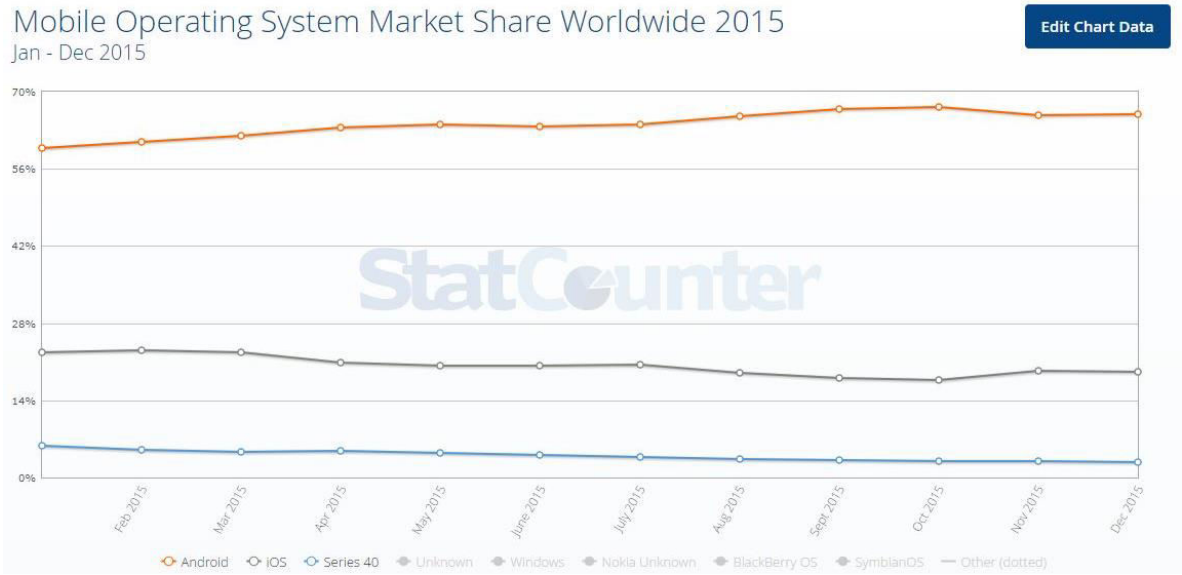

Figura 7. *Distribuição de uso de sistemas operacionais móveis em 2015.*

Fonte: (STATCOUNTER, 2015).

O duopólio Google/Android e Apple/iOS talvez tenha atingido seu ápice no último trimestre de 2016, quando, conforme ilustrado na tabela 1, mundialmente, 99,6% dos smartphones vendidos utilizavam uma das duas plataformas. Da porcentagem acima mencionada, o Android tem a maior parte, 81,7% enquanto o iOS representa 17,9% (VINCENT, 2017).

Sistema Operacional	Unidades 4º trim. 2016	Fatia de mercado 4º trim. 2016 (%)
Android	352,669.9	81.7
iOS	77,038.9	17.9
Windows	1,092.2	0.3
BlackBerry	207.9	0.0
Outros SOs	530.4	0.1
Total	**431,539.3**	**100**

Tabela 1. *Tabela com o número de smartphones vendidos no 4º trimestre de 2016.*

Fonte: (VINCENT, 2017).

Mundialmente, a distribuição atual não apresentou mudanças significativas e o Android segue como líder com uma grande margem de vantagem em relação ao iOS, como apresentado na figura 8.

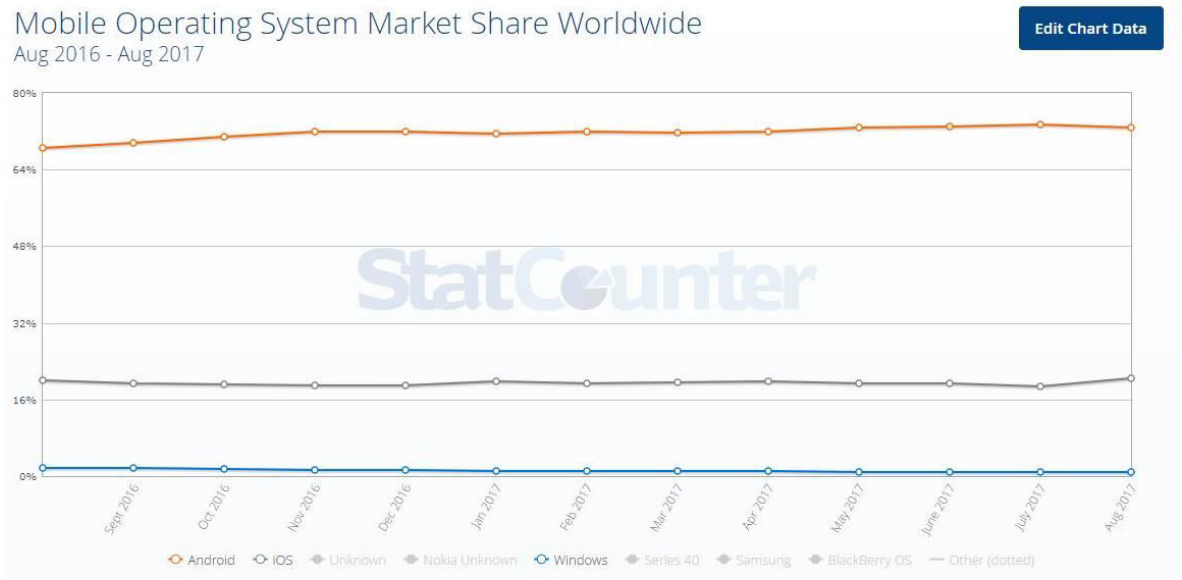

Figura 8. *Distribuição de uso de sistemas operacionais móveis entre agosto de 2016 e agosto de 2017.*

Fonte: (STATCOUNTER, 2017).

É preciso salientar que os gráficos apresentados nesta seção são de abrangência mundial, de modo que cada parte do mundo apresenta sua própria distribuição, e em algumas delas o Android é superado pelo o iOS, como nos Estados Unidos, cuja distribuição é apresentada na figura 9. Já no Brasil o Android tem grande supremacia de utilização, conforme ilustrado pela figura 10.

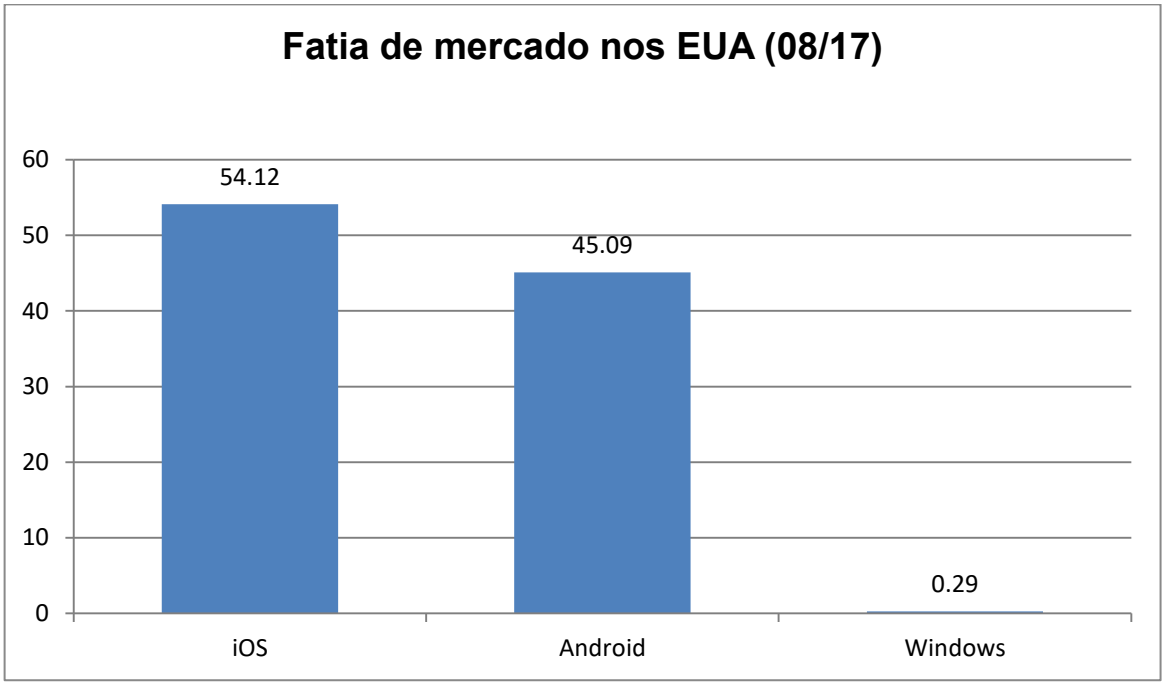

Figura 9. *Distribuição de uso de sistemas operacionais móveis nos EUA em agosto de 2017.*

. **Fonte:** (STATCOUNTER, 2017).

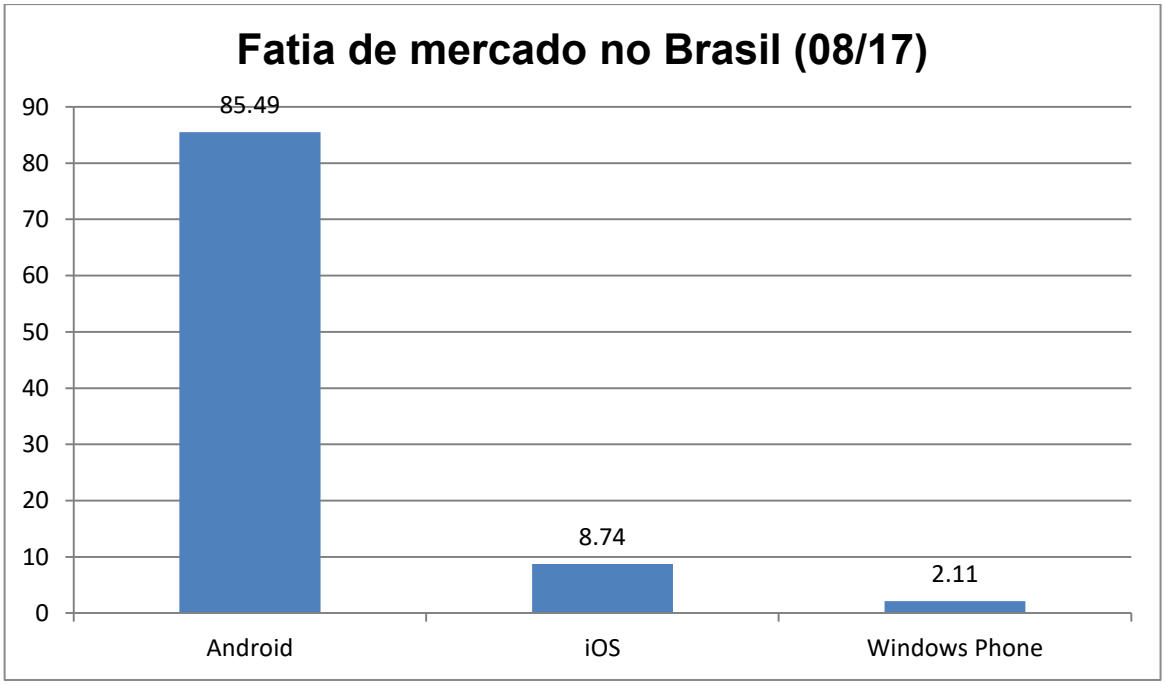

Figura 10. *Distribuição de uso de sistemas operacionais móveis no Brasil em agosto de 2017.*

Fonte: (STATCOUNTER, 2017).

Em março de 2017 um fato histórico ocorreu: o Android ultrapassou o Windows e tornou-se o sistema operacional mais popular da internet. O software da Microsoft era o líder desde a década de 1980 e a ascensão do Android demonstra a representatividade dos dispositivos móveis atualmente (G1, 2017). A distribuição em agosto de 2017 (englobando todas as categorias de sistemas operacionais) é apresentada na figura 11.

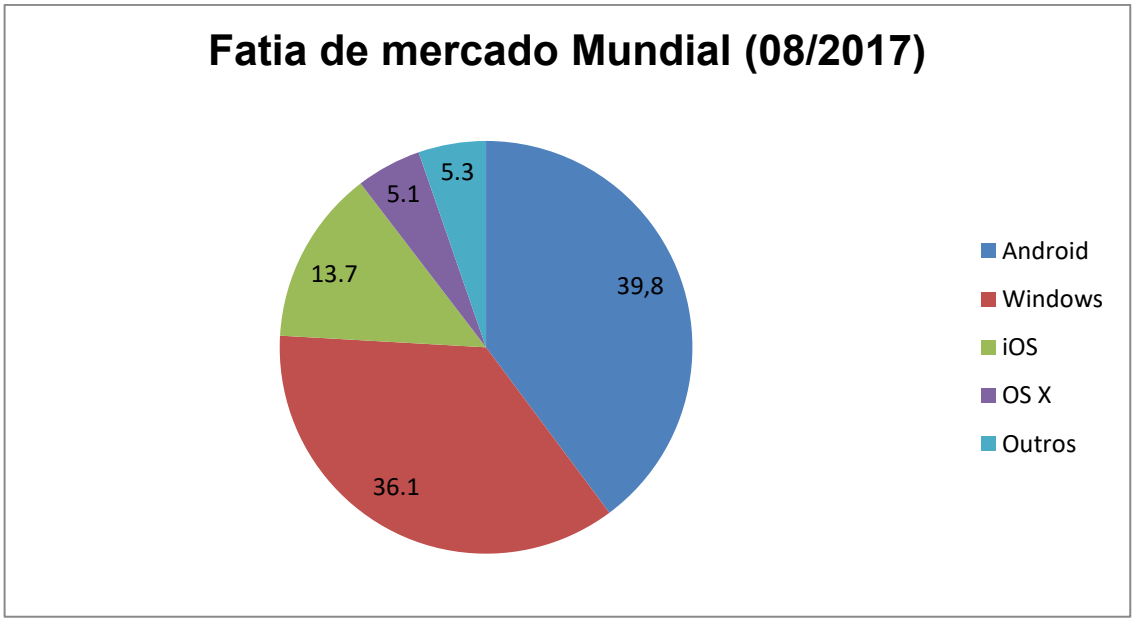

Figura 11. *Distribuição de uso de sistemas operacionais (não só móveis) em agosto de 2017.*

Fonte: (STATCOUNTER, 2017).

3.2. USABILIDADE E EXPERIÊNCIA DOS USUÁRIOS

A usabilidade está relacionada com a facilidade de aprendizado e uso da interface, bem como a satisfação do usuário em decorrência desse uso (Nielsen, 1993). Neste contexto, podemos caracterizar, resumidamente, a experiência de usuário (*user experience* ou *UX*) como o conjunto de experiências e emoções vivenciadas por ele ao fazer uso de um sistema. A experiência de usuário é associada aos estados dinâmicos de transição do sistema que são disparados por entradas do usuário. A qualidade da experiência de usuário é determinada por

resultados como a capacidade de resposta perceptível pelo usuário, suavidade, coerência e precisão. (LI, WANG, *et al.*, 2012)

Modelar a interface de um aplicativo para Android certamente exige mais do desenvolvedor do que modelar para o iOS. O Android é o sistema operacional presente em uma grande variedade de *smartphones* de diferentes fabricantes, como Motorola, LG, Samsung, HTC, Asus, Sony, entre outros, e, dessa forma, não só o sistema operacional, mas suas aplicações são executadas em telas com os mais variados tamanhos e resoluções. Isso acaba gerando um grande transtorno, já que dificilmente um desenvolvedor conseguirá fazer que sua aplicação apresente exatamente a mesma aparência em todos os dispositivos. Outro fator que contribui negativamente é que muitos dispositivos com sistema Android ainda fazem uso de versões antigas do sistema operacional, e tal situação dificulta ainda mais o trabalho dos desenvolvedores, uma vez que nem todos os aparelhos dispõem dos mesmos recursos. É importante enfatizar que tal situação não ocorre por opção dos usuários, mas pelo fato de que as atualizações do Android são geralmente lentas para chegar à maioria dos dispositivos. Para aparelhos que não estão sob marcas que utilizam o Android “puro” (isto é, sem alterações em relação ao software original), a atualização para a versão mais recente pode levar meses a partir da data de lançamento oficial. Um importante fator que contribui para este cenário é a grande variedade de *hardware* dos dispositivos Android, o que faz com que as atualizações devam passar por modificações específicas de acordo com cada aparelho (CUNNINGHAM, 2012). O site oficial do Android divulgou em setembro de 2017 os dados referentes à distribuição das versões de Android em atividade, apresentados na tabela 2.

Versão	Apelido	API	Distribuição
2.3.3 – 2.3.7	*Gingerbread*	10	0.6%
4.0.3 – 4.0.4	*Icre Cream Sandwich*	15	0.6%
4.1.x	*Jelly Bean*	16	2.4%
4.2.x	*Jelly Bean*	17	3.5%
4.3	*Jelly Bean*	18	1.0%
4.4	KitKat	19	15.1%
5.0	Lollipop	21	7.1%
5.1	Lollipop	22	21.7%
6.0	*Marshmallow*	23	32.2%
7.0	*Nougat*	24	14.2%
7.1	*Nougat*	25	1.6%

Tabela 2. *Distribuição de uso das versões do Android em agosto de 2017.*

Fonte: (ANDROID, 2017).

O iOS, por sua vez, não compartilha esse cenário; Por ser um sistema operacional *closed-source*, os dispositivos iOS não apresentam variações de diferentes fabricantes. A fabricação dos dispositivos móveis da Apple (iPhone, iPad e iPod) é quase completamente centralizada pela fabricante de eletrônicos Foxconn. Além disso, o iOS também não apresenta a heterogeneidade do Android no que tange à distribuição de usuários nas versões do sistema operacional. Dessa forma, é natural que o iOS seja uma referência em qualidade de interface e experiência de usuário.

3.3. DESENVOLVIMENTO DE APLICAÇÕES

Para desenvolver aplicações para dispositivos com o sistema Android, é necessário obter o SDK, que é disponibilizado gratuitamente.

O SDK permite aos desenvolvedores criarem aplicativos nativos para a plataforma Android. Ferramentas de desenvolvimento, emuladores, projetos simples

com código fonte e bibliotecas necessárias para criar as aplicações são alguns dos componentes contidos no Kit de Desenvolvimento para Android (CORDEIRO, 2017).

É possível utilizar as ferramentas do SDK através do *prompt* de comando, mas o Google recomenda a utilização de sua IDE (*Integrated Development Environment*) oficial, o Android Studio. É importante mencionar que outras IDEs possuem compatibilidade com o SDK, como o Visual Studio e o Eclipse, por exemplo.

O desenvolvimento de aplicações para dispositivos móveis é uma das áreas de maior destaque da computação atualmente. Neste contexto, há diferentes linguagens e ambientes de programação para os diversos sistemas operacionais móveis. Enquanto o Android utiliza a linguagem Java, o iOS faz uso de Objective-C e Swift, e o Windows Mobile de C#. Esta realidade acaba fazendo com que ao desenvolver uma aplicação, seja definida uma plataforma alvo, de modo a poupar tempo, custo e esforços durante seu desenvolvimento. No entanto, já existem *frameworks* que possibilitam o desenvolvimento multiplataforma. Talvez o mais famoso deles seja o *Xamarin*, que permite a utilização de C# para a criação de aplicações nativas que podem ser executadas em diferentes plataformas, como o iOS, Windows Mobile e Android. Outro importante *framework* nesse ramo é o *Apache Cordova* (antigo *Phonegap*). Por meio dele, é possível desenvolver aplicações móveis híbridas através de JavaScript, HTML e CSS.

3.4. ASSISTENTES PESSOAIS VIRTUAIS

Mesmo com os constantes aperfeiçoamentos de aplicativos e jogos para *smartphones*, certamente o que mais deslumbra os usuários destes dispositivos são as assistentes pessoais virtuais. Capazes de desempenhar um vasto conjunto de tarefas, que vai desde realizar chamadas telefônicas a contar uma piada, elas se popularizaram de forma maciça desde que surgiram. Tal cenário propiciou o surgimento de uma extensa lista de assistentes virtuais que inclui – entre as principais - Siri (Apple), Cortana (Windows), Google Now (Android) e Bixby (Samsung), que, dentre estas, é a mais recente (inclusive ainda sem suporte ao idioma português).

Dentre as mencionadas, a primeira a ser lançada foi a Siri. A assistente da Apple já está há algum tempo no mercado desde seu lançamento em 2011, em conjunto com o iOS 5. Em virtude de seu sucesso, ela passou por inúmeras atualizações e aprimoramentos, e passou a ser um dos principais diferenciais dos produtos Apple. Utilizando complexos conceitos de aprendizado de máquina, redes neurais e inteligência artificial, conforme vai interagindo com um determinado usuário, mais a Siri aprende sobre ele: seus lugares favoritos, preferências e tendências, de modo a se tornar personalizada para cada um. Para uma mesma pergunta e/ou solicitação, a Siri pode gerar respostas diferentes para diferentes usuários. Também é conhecida por ser uma fiel amiga virtual, dotada de personalidade e senso de humor, ela não só facilita a vida do usuário realizando tarefas como é capaz de responder perguntas informais e estabelecer um diálogo coloquial.

O Google Now está disponível desde o Android 4.1 (*Jelly Bean*) e é ativado por voz, assim como outras assistentes pessoais virtuais, mas ao contrário das demais, o Google Now funciona em mais de uma plataforma: é possível utilizá-lo tanto no Android quanto no iOS, embora existam algumas limitações ao ser utilizado no sistema operacional da Apple. Assim como os concorrentes, ele permite a realização de diversas tarefas, como agendar eventos e alarmes, ajustar o volume do dispositivo e postar nas redes sociais através de comandos de voz.

O Google também possui uma segunda assistente virtual: O Google Assistant. Anunciada em 2016, esta, por sua vez, não opera por meio de comandos de voz. Ela inicialmente estreou como parte do aplicativo de mensagens Google Allo, sendo possível fazer perguntas e solicitações a assistente virtual. O Google Assistant vem com o objetivo de propiciar uma experiência mais amigável com o usuário, no sentido de não apenas realizar tarefas para ele (como o Google Now) mas também estabelecer diálogos mundanos. Inicialmente implantada apenas nos dispositivos Google Pixel, desde fevereiro de 2017 já está sendo disponibilizada para certos dispositivos com Android (PUREWAL, 2016).

3.5. INTEGRAÇÃO COM OUTROS DISPOSITIVOS

Uma das principais características do Android é a abrangente gama de dispositivos que o utilizam. Isso se deve a sua natureza *open source* e altamente personalizável.

Há muito tempo o Android ultrapassou a fronteira *smartphone*/*tablet* e já está presente em dispositivos como fones de ouvido, *smartwatches*, *smart* TVs, tocadores de CD e DVD de carros, espelhos, telefones fixos, MP3s, videogames e outros aparelhos (PETROVAN, 2012).

Já existe uma variedade de relógios inteligentes (*smartwatches*) no mercado; Motorola, ASUS, Sony e LG são apenas algumas das empresas que já tem o seu próprio dispositivo. Relógios e pulseiras inteligentes têm como sistema operacional o *Android Wear*, cuja principal diferença em relação ao Android é o fato de o Google não permitir a personalização da interface como acontece no Android (CARRARA, 2014). Com suporte a comandos por movimento e voz (através do Google Now), os *smartwatches* podem contar com uma série de recursos como sensores de movimento e luz, monitores cardíacos, sensores de GPS, entre outros, além de fornecerem ao seu usuário certa independência em relação ao *smartphone*, oferecendo acesso a alguns aplicativos e informações sem a necessidade de utilizar o celular (RINALDI, 2015).

O Android inclusive já está presente em vários videogames. Os destaques talvez fiquem por conta do *Shield Portable* da Nvidia e o Ouya. Este último inclusive foi um grande sucesso da *Kickstarter* (empresa que mantém uma plataforma de financiamento coletivo) que arrecadou por meio de doações a expressiva marca de U$8,5 milhões para o desenvolvimento do console (LIEN, 2012).

O Android também já é empregado em espelhos e balanças, sendo possível enviar mensagens e verificar a previsão do tempo através de interações com o espelho e também analisar gráficos que mostram a variação do peso do usuário ao longo das últimas semanas (PETROVAN, 2012).

Ao que parece, enquanto houver desenvolvedores dispostos a adaptar o Android para novos dispositivos, o sistema operacional vai ser sempre capaz de surpreender a todos com sua adaptabilidade.

4. O SISTEMA OPERACIONAL ANDROID

O presente capítulo certamente é o principal deste trabalho. Permeando a maior parte dos conceitos fundamentais associados aos sistemas operacionais como gerenciamento de memória e processos, sistema de arquivos, arquitetura, segurança, entre outros, o capítulo quatro propõe-se a analisar de forma abrangente o sistema Android e traçar um paralelo com o Linux de forma que fique claro o que ambos têm em comum e o que o Android apresenta de novidade.

4.1. ARQUITETURA E ESTRUTURA DO SISTEMA

A estrutura do Android é organizada em cinco camadas com seis seções (desconsiderando a camada de Gerenciamento de Energia). Cada camada mantém forte integração com seus elementos internos e provê diferentes tipos de serviço a camada superior. A figura 12 ilustra tal modelo:

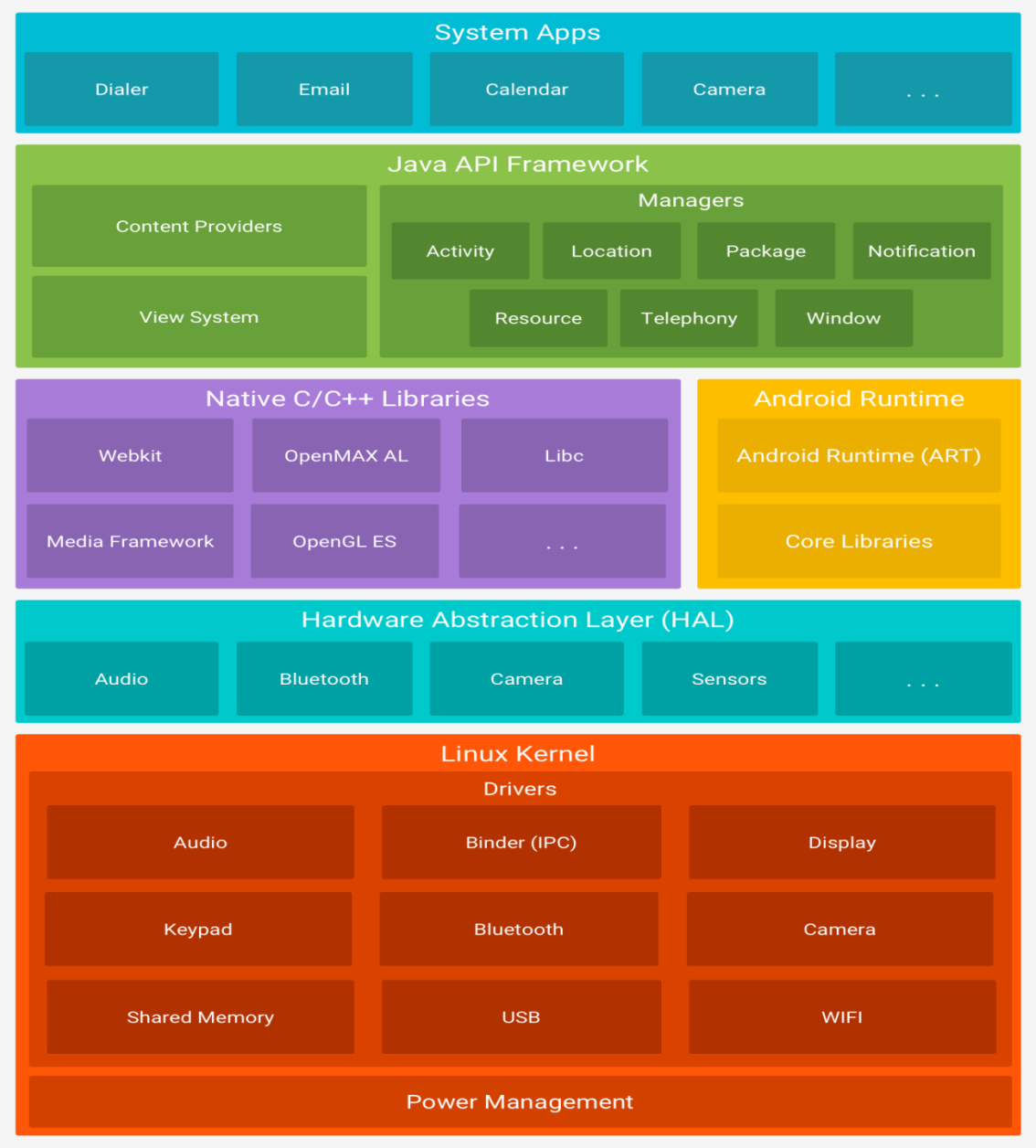

Figura 12. *A arquitetura do Android.*

Fonte: (ANDROID, 2017).

4.1.1. O KERNEL LINUX

A camada do *kernel* Linux além de ser a responsável por prover os principais serviços do sistema operacional, também realiza a abstração entre o *hardware* e os aplicativos.

Apesar de o Android utilizar várias funcionalidades do *kernel* Linux, algumas alterações foram realizadas de modo a adequá-lo a sua nova realidade - executar

em dispositivos móveis, que apresentam consideráveis diferenças quando comparados aos sistemas computacionais tradicionais. Dessa forma, o *kernel* do Android apresenta inúmeras modificações, entre elas: novos *drivers* de dispositivos, sistema de gerenciamento de energia e um Matador de Falta de Memória (que será abordado na seção 4.5).

Outros recursos exclusivos do *kernel* do Android incluem, entre os principais, o *Binder*, que é o mecanismo para comunicação interprocesso, o *Ashmem*, um novo mecanismo de compartilhamento de memória que possibilita a comunicação entre dois ou mais processos em uma região compartilhada de memória e os *Wakelocks* (GOMES, FERNANDES e FERREIRA, 2012), que tratam da parte de gerenciamento de energia, que será avaliada na seção 4.7.

4.1.2. A CAMADA DE ABSTRAÇÃO DE HARDWARE

Acima da camada do *kernel* localiza-se a camada de abstração de hardware (HAL). A HAL consiste em módulos de biblioteca, que implementam uma interface para um tipo específico de componente de *hardware*, como o módulo de câmera ou *bluetooth*. Quando um *Framework* API faz uma chamada para acessar o *hardware* do dispositivo, o sistema Android carrega o módulo da biblioteca para este componente de *hardware* (ANDROID, 2017).

4.1.3. BIBLIOTECAS NATIVAS C/C++

Acima da *HAL* ficam as bibliotecas (em geral desenvolvidas em C ou C++) que são utilizadas por diversos componentes do sistema. Entre as principais, pode-se mencionar: a OpenGL/ES para lidar com a interface gráfica, a SQLite para trabalhar com banco de dados, o motor de renderização de páginas web para navegadores WebKit e o SSL (*Secure Sockets Layer)* que fornece encriptação de dados enviados pela *internet*, além do *surface manager,* que compõe as imagens em 2D e 3D e também controla e gerencia o acesso ao subsistema de *display* (BORDIN, 2012).

4.1.4. ANDROID RUNTIME

As aplicações escritas em Java executam em sua própria máquina virtual, que não é a JVM (Java *Virtual Machine)*, mas a ART (*Android Runtime),* especialmente projetada para o Android sobre a qual as aplicações e boa parte dos processos do sistema executam, isolando cada aplicação e seu respectivo processo uns dos outros, visando promover um maior nível de segurança. Na camada do *Android Runtime* há um conjunto de bibliotecas que fornece boa parte das funcionalidades encontradas nas bibliotecas padrão do Java. O ART passou oficialmente a ser a máquina virtual do Android a partir da versão 5.0 do sistema. Dispositivos com versões anteriores a essa tem como máquina virtual o Dalvik. Tanto o ART como o Dalvik desempenham as mesmas funções, no entanto, o ART apresenta algumas vantagens em relação ao seu antecessor, entre elas, coleta de lixo aprimorada e a compilação "*ahead-of-time*" (AOT) que produz código de máquina ainda mais otimizado (ANDROID, 2017) (YADAV e MAHESHWARI, 2016).

4.1.5. FRAMEWORK DE APLICAÇÕES

Acima das camadas *Android Runtime* e Bibliotecas Nativas C/C++, aparece a camada de *framework* Java API, que pode ser também chamada de *Framework* de Aplicações. Essa camada disponibiliza aos desenvolvedores as mesmas APIs utilizadas para a criação de aplicações nativas do Android. Essa camada abstrai a complexidade e simplifica a reutilização de componentes. Os elementos presentes nessa camada são:

- *Activity Manager* (Gerenciador de atividades): Gerencia o ciclo de vida das aplicações;
- *Location Manager* (Gerenciador de localização): Gerencia a localização do dispositivo;
- *Package Manager* (Gerenciador de pacotes): Responsável por manter o registro de todas as aplicações instaladas no dispositivo;

- *Notification Manager* (Gerenciador de notificações): Permite que as aplicações notifiquem o usuário sobre um evento através de som, vibração ou alertas na barra de status;
- *Resource Manager* (Gerenciador de recursos): Fornece acesso a recursos gráficos e arquivos de *layout;*
- *Telephony Manager* (Gerenciador de telefonia): Fornece acesso a recursos de telefonia;
- *Window Manager* (Gerenciador de janelas): Gerencia as janelas das aplicações;
- *Content Providers* (Provedores de conteúdo): Fornece compartilhamento de dados entre aplicações;
- *View System* (Visões do sistema): Fornece um conjunto básico de *views* que podem ser utilizadas por aplicações, como: listas, caixas de texto, botões, etc (AQUINO, 2007).

4.1.6. APLICATIVOS DO SISTEMA

A camada mais externa é chamada de ‘Aplicativos do Sistema’, que abrange os aplicativos nativos da versão em execução do Android (assim como os que serão instalados posteriormente). Alguns exemplos são o cliente de e-mail, relógio, calendário, câmera, entre outros.

4.2. APLICAÇÕES PARA ANDROID

Segundo Tanenbaum & Bos (2016), uma aplicação Android é um arquivo com uma extensão *apk,* de *Android Package*. Esse arquivo na realidade é um arquivo compactado regular, contendo tudo sobre a aplicação. Os conteúdos mais importantes de um *apk* são:

1. Um manifesto descrevendo o que é a aplicação, o que ela faz e como executá-la. O manifesto deve fornecer um nome de *package* para a

aplicação, uma cadeia de caracteres com escopo no estilo Java (como *com.android.app.calculator*), que a identifique unicamente.

2. Os recursos necessários pela aplicação, incluindo cadeias de caracteres que são exibidos para o usuário, dados XML para *layouts* e outras descrições, mapas de bits gráficos, etc.
3. O código em si, que pode ser o *bytecode* Dalvik*,* assim como o código nativo de bibliotecas.
4. Informações de assinatura, identificando com segurança o autor.

A parte fundamental da aplicação é o seu manifesto, que é um arquivo XML pré-compilado chamado AndroidManifest.xml na raiz do espaço de nomes *zip* do *apk*. Um exemplo completo de declaração de manifesto para uma aplicação hipotética de *e-mail* é apresentada na figura 12. Pode-se observar que ele permite a visualização e composição de e-mails e também inclui componentes necessários para sincronizar seu armazenamento local de *e-mails* com um servidor mesmo quando o usuário não esteja na aplicação.

Segundo Tanenbaum & Bos (2016), as aplicações Android não têm um ponto de entrada *main* simples que seja executado quando o usuário as inicia. Em vez disso, as aplicações publicam sob a etiqueta do manifesto <*application*>, através de vários pontos de entrada descrevendo o conjunto de funcionalidades que a aplicação pode desempenhar. Esses pontos de entrada são expressos como quatro tipos distintos, definindo os tipos centrais de comportamento que as aplicações podem fornecer: atividade, receptor, serviço e provedor de conteúdo. A figura 13 ilustra algumas atividades e uma declaração dos outros tipos de componentes, mas uma aplicação pode declarar zero ou mais de qualquer um destes.

Cada um dos quatro tipos de componentes que uma aplicação pode conter tem diferentes semânticas e usos dentro do sistema. Em todos os casos, o atributo *android:name* fornece o nome de uma classe Java do código de aplicação implementando aquele componente, que será instanciado pelo sistema quando necessário.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.example.email">
   <application>

      <activity android:name="com.example.email.MailMainActivity">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

      <activity android:name="com.example.email.ComposeActivity">
         <intent-filter>
            <action android:name="android.intent.action.SEND" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType="*/*" />
         </intent-filter>
      </activity>

      <service android:name="com.example.email.SyncService">
      </service>

      <receiver android:name="com.example.email.SyncControlReceiver">
         <intent-filter>
            <action android:name="android.intent.action.DEVICE_STORAGE_LOW" />
         </intent-filter>
         <intent-filter>
            <action android:name="android.intent.action.DEVICE_STORAGE_OKAY" />
         </intent-filter>
      </receiver>

      <provider android:name="com.example.email.EmailProvider"
            android:authorities="com.example.email.provider.email">
      </provider>

   </application>
</manifest>
```

Figura 13. *Manifesto de uma aplicação Android.*

Fonte: (TANENBAUM e BOS, 2016).

De acordo com Tanenbaum & Bos (2016), o gerenciador de pacotes é a parte do Android que controla todos os pacotes de aplicação. Ele analisa cada manifesto das aplicações, coletando e indexando as informações que encontra nelas. Com essas informações, ele proporciona facilidade para os usuários questionarem sobre as aplicações atualmente instaladas e recuperar informações relevantes sobre elas. Ele também é responsável por instalar aplicações (criando espaço de armazenamento para a aplicação e assegurando a integridade do *apk*) assim como tudo o que é necessário para desinstalar (limpar tudo que seja associado ao aplicativo previamente instalado).

Outro serviço importante do sistema é o gerenciador de atividades. Enquanto o gerenciador de pacotes é responsável por manter a informação estática

a respeito de todas as aplicações instaladas, o gerenciador de atividades determina quando, onde e como essas aplicações devem ser executadas. Apesar do nome, ele na realidade é responsável por executar todos os quatro tipos de componentes de aplicação e implementar o comportamento adequado para cada um deles.

4.2.1. ATIVIDADES

Segundo Tanenbaum & Bos (2016), uma atividade é uma parte da aplicação que interage diretamente com o usuário por meio de uma interface. Quando um usuário executa uma aplicação em um dispositivo, isso é na realidade uma atividade dentro da aplicação que foi designada como um ponto de entrada principal.

O manifesto de e-mail apresentado na figura 13 contém duas atividades. A primeira é a principal interface do usuário de e-mail, que mostra aos usuários suas mensagens; a segunda é uma interface separada para compor uma nova mensagem. A primeira atividade de e-mail é declarada como o principal ponto de entrada para a aplicação, isto é, a atividade que será inicializada quando o usuário executar o aplicativo.

O sistema apresentará o estado ilustrado pela figura 14 quando for executado o aplicativo de e-mail.

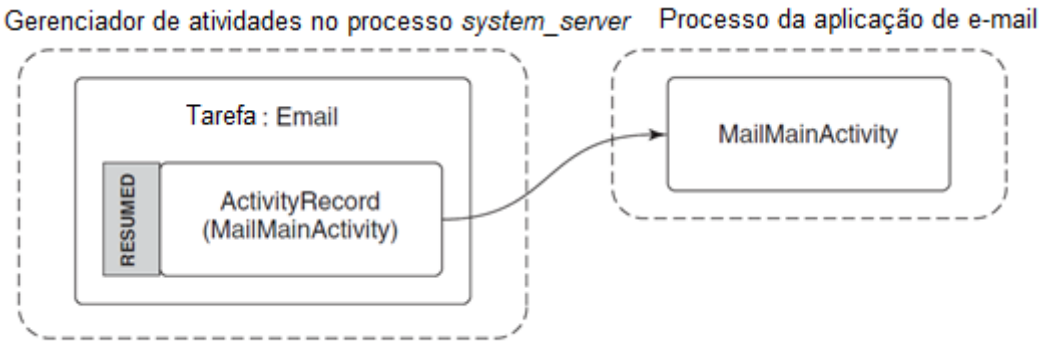

Figura 14. *Começando uma atividade principal de uma aplicação de e-mail.*

Fonte: (TANENBAUM e BOS, 2016).

Neste ponto, o gerenciador de atividades criou uma instância de *ActivityRecord* interna em seu processo para controlar a atividade. Uma ou mais dessas atividades são organizadas em containers chamados tarefas (*tasks*). Nesse ponto, o gerenciador de atividades começou o processo de aplicação do e-mail e uma instância do seu *MailMainActivity* para exibir sua interface de usuário principal. Essa atividade está em um estado de retomada (*resumed*), uma vez que ela está no primeiro plano da interface do usuário.

Suponha que o usuário troque o aplicativo de e-mail (sem fechá-lo) pelo aplicativo da câmera para tirar uma foto, o estado do sistema estaria de acordo com a figura 15.

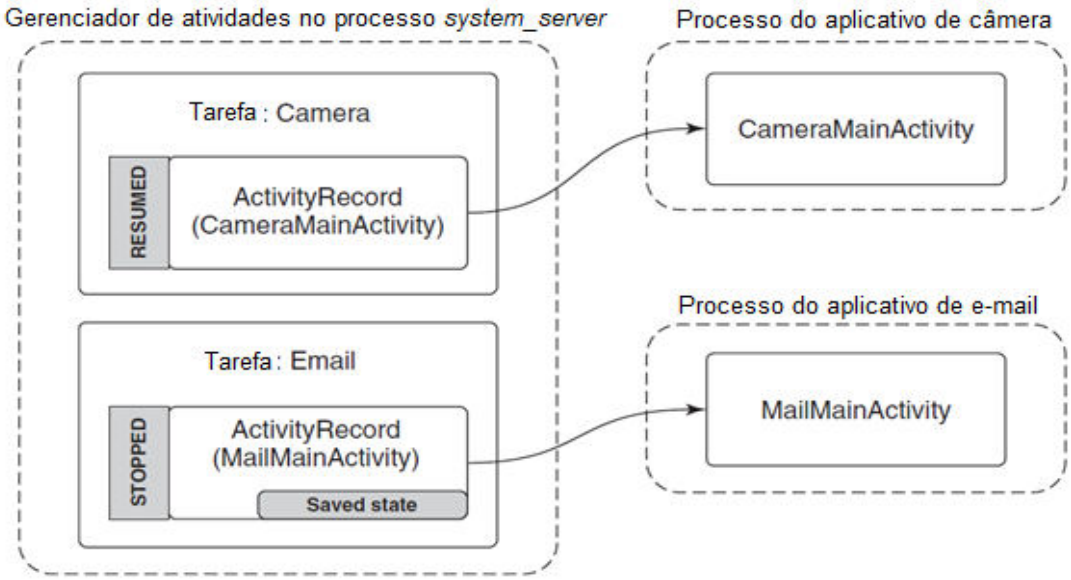

Figura 15. *Começando a aplicação de câmera após ter aberto a de e-mail.*

Fonte: (TANENBAUM e BOS, 2016).

Agora temos um novo processo de câmera executando a principal atividade da câmera, um *ActivityRecord* associado a ele no gerenciador de atividades e agora esta é a atividade retomada. Já a atividade de e-mail anterior passa do estado de retomado para "parado" e seu *ActivityRecord* armazena seu estado salvo.

Assim que uma atividade deixa de estar em primeiro plano, o sistema pede a ela para "salvar seu estado". Isso faz com que a aplicação crie uma quantidade mínima de informação do estado representando o que o usuário vê atualmente, que ela retorna ao gerenciador de atividades e armazena no processo *system_server*, no *ActivityRecord* associado a aquela atividade. O estado salvo para uma atividade geralmente é pequeno, contendo, por exemplo, onde você se encontra em uma mensagem de *e-mail*, mas não a mensagem em si, que será armazenada em outra parte pelo aplicativo em seu armazenamento persistente.

Suponha agora que o aplicativo de câmera passou a exigir muita memória principal, sobrecarregando o sistema. Ele então pode simplesmente, através do Matador de Falta de Memória (*Low Memory Killer,* que será explicado na seção 4.5), ceifar o processo de *e-mail*, como mostra a figura 16. No entanto, o *ActivityRecord* associado ao processo de *e-mail* segue com seu estado salvo, seguramente “escondido” pelo gerenciador de atividades no processo *system_server.* Tendo em

vista que o processo do *system_server* hospeda todos os serviços centrais do sistema Android, ele deve seguir sempre executando, de maneira que o estado salvo aqui permanecerá por tanto tempo quanto for necessário.

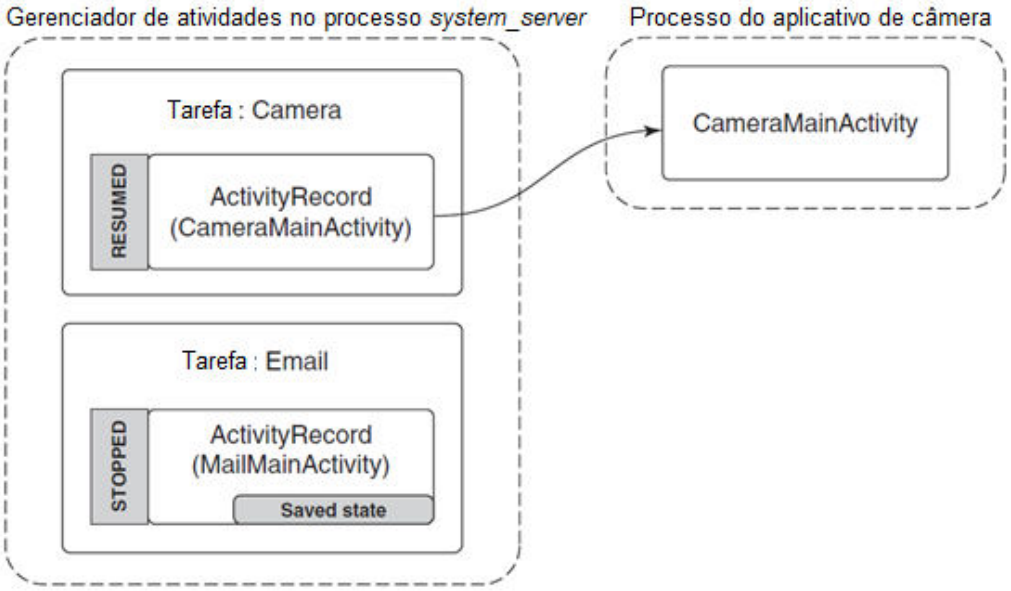

Figura 16. *Encerrando o processo de e-mail para liberar memória principal para a câmera.*

Fonte: (TANENBAUM e BOS, 2016).

O aplicativo hipotético de *e-mail* possui outra atividade além da que fornece a interface de usuário principal, é a *ComposeActivity*. As aplicações podem declarar quantas atividades quiserem, elas são importantes, pois participam do sistema de compartilhamento entre as aplicações do Android. Suponha que o usuário, no aplicativo da câmera, decida compartilhar uma foto que ele tirou, a *ComposeActivity* do aplicativo de e-mail é uma das opções de compartilhamento que ele tem. Caso essa seja a opção escolhida, a *ComposeActivity* será inicializada e lhe será entregue a foto a ser compartilhada. O novo estado do sistema está representado pela figura 17.

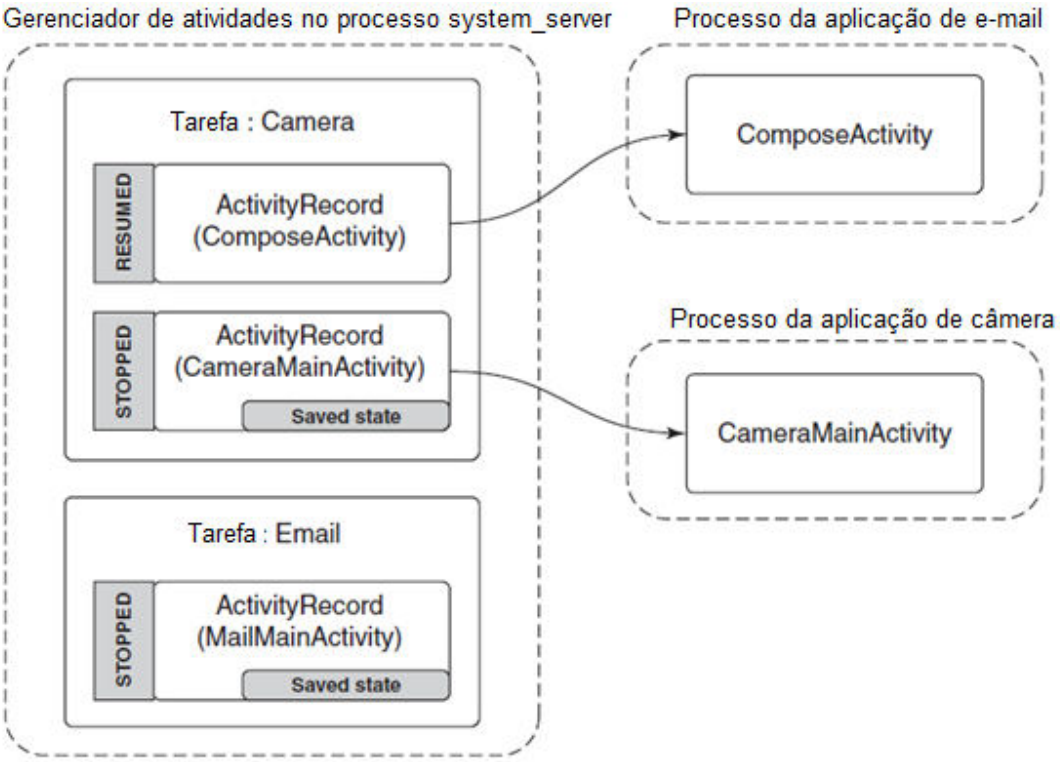

Figura 17. *Compartilhando uma foto da câmera através da aplicação de e-mail*.

Fonte: (TANENBAUM e BOS, 2016).

Há uma série de questões importantes a serem observadas:

1. O processo do aplicativo de e-mail deve ser inicializado novamente, para executar a sua *ComposeActivity*.
2. No entanto, o antigo *MailMainActivity* não é inicializado, uma vez que ele não é necessário. Isso reduz o uso da RAM.
3. A tarefa da câmera agora tem dois registros: o original *CameraMainActivity* no qual o usuário acabou de estar, e a nova *ComposeActivity,* que está agora em exibição. Para o usuário, essas são ainda uma tarefa coesa: trata-se da câmera atualmente interagindo com eles para enviar uma foto por *e-mail*.

4. A nova *ComposeActivity* está no topo, então ela é retomada; o *CameraMainActivity* anterior não está mais no topo, então o seu estado foi salvo. Podemos neste ponto seguramente sair deste processo se essa RAM for necessária em outra parte.

4.2.2. SERVIÇOS

De acordo com a definição apresentada por Tanenbaum & Bos (2016), um serviço tem duas identidades distintas:

1. Pode ser uma operação autocontida de segundo plano de longa execução. Um exemplo comum da utilização de serviços dessa maneira é produzir uma música em segundo plano enquanto o usuário está em outros aplicativos, baixar ou enviar dados em segundo plano, etc.
2. Ele pode servir como um ponto de conexão para outros aplicativos ou o sistema para desempenhar uma interação rica com o aplicativo. Isso pode ser usado por aplicativos para fornecer APIs seguras para outros aplicativos, como realizar processamento de imagem ou áudio, fornecer um texto para fala, etc.

O exemplo do manifesto de *e-mail* representado pela figura 13 contém um serviço que é usado para realizar a sincronização da caixa de *e-mail* do usuário. Uma implementação comum escalonaria o serviço em intervalos regulares, como a cada 15 minutos. Esse é o uso típico da primeira categoria de serviço, uma longa operação de segundo plano.

4.2.3. RECEPTORES

A função de um receptor é receber os eventos (tipicamente externos) que acontecem geralmente no segundo plano e fora da interação do usuário normal.

Conceitualmente, receptores fazem o mesmo que um aplicativo registrando para um *call-back* quando algo interessante acontece (um alarme dispara, mudanças na conectividade de dados, entre outros), mas não exigem que a aplicação esteja executando a fim de receber o evento.

O exemplo de manifesto de *e-mail* apresentado anteriormente contém um receptor para o aplicativo descobrir quando a capacidade de armazenamento do dispositivo se torna baixa para que ele pare de sincronizar o *e-mail* (pois trata-se de uma tarefa que pode consumir mais espaço de armazenamento).

Quando a capacidade de armazenamento do dispositivo torna-se baixa, o sistema enviará uma transmissão a todos (*broadcast*) com o código referente ao

status de baixa capacidade de armazenamento, para ser entregue a todos os receptores interessados no evento.

A figura 18 ilustra como uma transmissão desta natureza é processada pelo gerenciador de atividades a fim de entregá-la aos receptores interessados. Primeiro é solicitado ao gerenciador de pacotes por uma lista de todos os receptores interessados no evento, que é colocado em um *Broadcast-Record* representando aquela transmissão. O gerenciador de atividades procederá então para passar por cada entrada na lista, fazendo com que cada processo do aplicativo associado crie e execute a classe de receptor adequada.

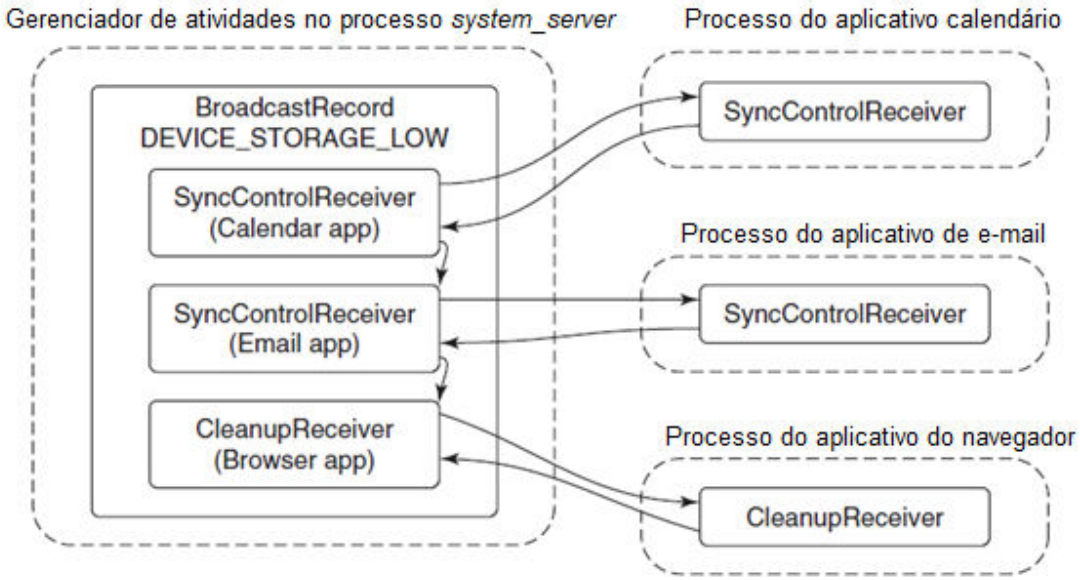

Figura 18. *Enviando um broadcast para receptores de aplicações.*
Fonte: (TANENBAUM e BOS, 2016).

4.2.4. PROVEDORES DE CONTEÚDO

O último componente de aplicação, o provedor de conteúdo, é um mecanismo fundamental que as aplicações usam para trocar dados entre si. Todas as interações com o provedor de conteúdo são através de URIs[1] (*Uniform Resource*

[1] Uma URI pode ser classificada como um localizador, um nome ou ambos. Uma URI, além de identificar um recurso, pode fornecer meios para localizar o recurso descrevendo seu mecanismo de acesso primário (por exemplo, sua "localização" no armazenamento persistente de um dispositivo).

Identifier) usando um conteúdo (*content)*. A autoridade do URI é usada para descobrir a implementação de provedor de conteúdo certa para interagir.

Por exemplo, na aplicação de e-mail hipotética, o provedor de conteúdo especifica que sua autoridade é *com.example.email.provider.email.* Desse modo, URIs operando nesse provedor de conteúdo começariam com: content:*//com.example.email.provider.email/messages.*

O sufixo para aquela URI é interpretado pelo próprio provedor para determinar quais dados dentro dele estão sendo acessados. Neste exemplo, uma convenção comum seria que a URI content*://com.example.email.provider.email/messages* significa a lista de todas as mensagens de e-mail, enquanto content:*//com.example.email.provider.email/messages/1* fornece acesso a uma única mensagem (a primeira).

Para interagir com um provedor de conteúdo, as aplicações sempre passam por uma API de sistema chamada *ContentResolver*, em que a maioria dos métodos tem um argumento de URI inicial indicando os dados para operar. Um dos métodos de *ContentResolver* mais comumente usados é *query;* que realiza uma consulta de banco de dados de uma determinada URI. Por exemplo, para recuperar um resumo de todas as mensagens do e-mail disponíveis se pareceria com algo como:

query(“content:*//com.example.email.provider.email/messages*”).

4.2.5. INTENTO

Um detalhe ainda não discutido no manifesto de aplicação mostrado acima são as etiquetas <*intent-filter*>, incluídas com as atividades e declarações do receptor. Elas fazem parte da característica de intento no Android, que é o mecanismo usado pelo sistema para descobrir e identificar atividades, receptores e serviços, parte fundamental para definir como aplicações diferentes identificam umas às outras a fim de serem capazes de interagir e trabalhar juntas.

Há dois tipos principais de intentos: explícito e implícito. Um intento explícito é aquele que identifica diretamente um único componente de aplicação específico. A parte mais importante de um intento desta categoria é um par de cadeias de caracteres nomeando o componente: o *package name* da aplicação-alvo e classe

name do componente dentro da aplicação. Agora, retornando ao exemplo da Figura 13 na aplicação do manifesto, um intento explícito para esse componente seria um com o nome de pacote *com.example.email* e nome de classe *com.example.email.MailMainActivity*.

O nome de classe e pacote de um intento explícito são informações suficientes para identificar unicamente um componente-alvo, como a atividade de *e-mail* principal no manifesto. A partir do nome do pacote, o gerenciador de pacotes pode retornar tudo o que for necessário a respeito da aplicação, como onde encontrar o seu código. A partir do nome de classe, sabemos qual parte daquele código executar.

Um intento implícito é aquele que descreve características do componente desejado, mas não o componente em si; Esse processo de encontrar o componente que pareie com um intento implícito é chamado de resolução de intento.

O mecanismo de compartilhamento geral do Android, apresentado na figura 17, é um bom exemplo de intentos implícitos. Neste ponto, a aplicação da câmera constrói um intento descrevendo a ação a ser feita, e o sistema encontra todas as atividades que têm potencial para realizar aquela ação. Um compartilhamento é solicitado por meio da ação de intento *android.intent.action.SEND* e podemos ver no manifesto que a atividade *Compose* declara que pode realizar essa ação.

Pode haver três resultados de uma resolução de intento: (1) nenhum pareamento é encontrado, (2) um único pareamento é encontrado ou (3) há múltiplas atividades que podem lidar com o intento. Um pareamento vazio dará um resultado vazio ou uma exceção, dependendo das expectativas do chamador a essa altura. Se o pareamento for único, então o sistema pode imediatamente proceder para lançar agora o intento explícito. Se o pareamento não for único, precisamos de alguma maneira solucioná-lo de outro modo para um único resultado.

4.3. PROCESSOS

O primeiro processo do espaço do usuário no Android (assim como no Linux) é o *init*, que é a raiz de todos os processos. O *init* inicializa um conjunto de

daemons [2]que se encarrega de detalhes de baixo nível (gerenciamento de sistemas de arquivos e acesso ao hardware). O Android também tem uma camada adicional de processos, aqueles executando sobre a máquina virtual ART (Android *Runtime*), que são responsáveis por executar todas as partes do sistema implementadas em Java.

A figura 19 ilustra a estrutura básica de processos do Android. O primeiro processo criado é o *init*, que gera processos de *daemon* de baixo nível. O *zygote* é um deles, que é a raiz dos processos de linguagem Java de nível mais alto.

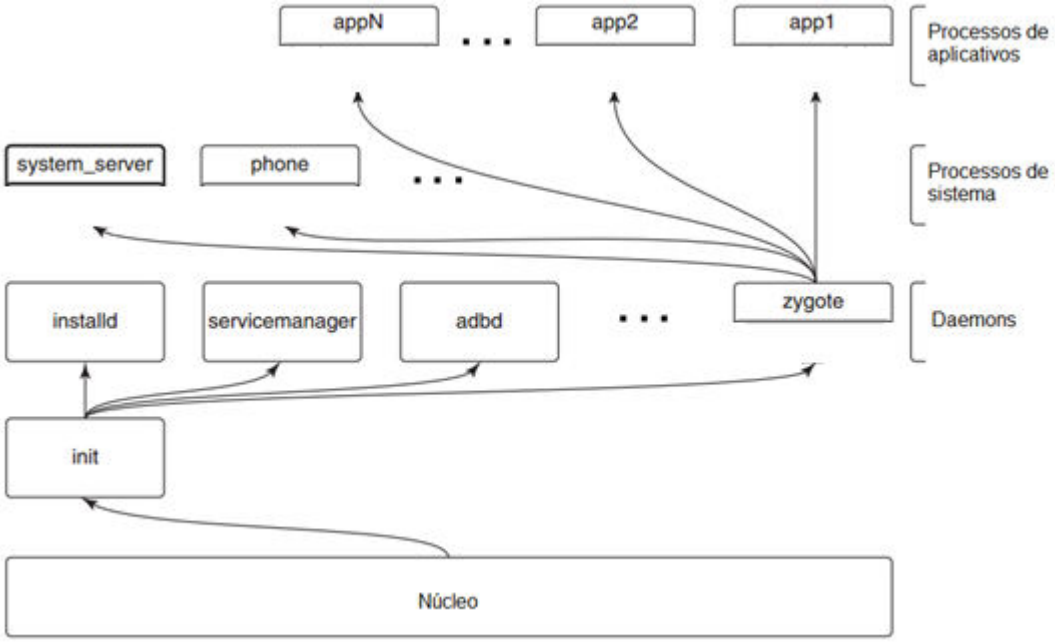

Figura 19. *Hierarquia de processos do Android.*
Fonte: (TANENBAUM e BOS, 2016).

Considerando que a maior parte do Android é escrita em Java, o *daemon zygote* e os processos inicializados a partir dele são centrais para o sistema. O processo *system_server* é sempre o primeiro a ser inicializado pelo *zygote*, contendo todos os serviços de base do sistema operacional. Os principais são os gerenciadores de energia, pacotes, janelas e atividades.

[2] *Daemons* são processos em segundo plano sem interação direta com o usuário.

Outros processos serão criados a partir do *zygote* conforme necessário. Alguns deles são persistentes, ou seja, devem permanecer sempre em execução, como a pilha de telefonia. Processos de aplicativos adicionais serão criados e parados conforme a necessidade enquanto o sistema estiver executando. O *zygote* também é responsável por carregar e inicializar a ART e todos os novos processos baseados nele (sistema ou aplicação).

Criar um processo a partir do *zygote* faz com que este seja uma réplica do *zygote* original, com todo o seu estado pré-inicializado já configurado e pronto. Após o *fork* (chamada de sistema para duplicar o processo *zygote* original), o novo processo tem o seu próprio ambiente ART separado, embora ele esteja compartilhando todos os dados pré-carregados e inicializados com o *zygote*. Só o que falta para ter o novo processo em execução pronto para executar é dar a ele sua identidade correta (UID, abordada na seção 4.6 de Segurança), terminar qualquer inicialização do ART que requer iniciar *threads* e carregar a aplicação ou código de sistema a ser executado (TANENBAUM e BOS, 2016).

4.3.1. A MÁQUINA VIRTUAL ART

A ART implementa o ambiente da linguagem Java no Android que é responsável por executar aplicações assim como a maior parte do código do sistema.

É preciso ressaltar que o Android não é uma plataforma Java tradicional. O código Java em uma aplicação Android é fornecido no formato *bytecode* do Dalvik em vez do formato *bytecode* do Java, isso possibilita uma interpretação mais veloz, já que há suporte a compilação *Just In Time* (JIT) e *Ahead Of Time* (AOT).

Ao desenvolver aplicações para Android utilizando Java, o código fonte é compilado em *bytecode* Java através de suas ferramentas tradicionais. O Android então introduz um novo passo: converter o *bytecode* Java em uma representação de *bytecode* mais compacta do Dalvik. Então, essa é a versão *bytecode* de uma aplicação que é colocada em um pacote como binário de uma aplicação final que é instalada no dispositivo (TANENBAUM e BOS, 2016).

Substituindo Dalvik, que é a máquina virtual de processos originalmente usada pelo Android, a ART (oficialmente nos dispositivos na versão 5.0) executa a

tradução do *bytecode* do aplicativo para instruções nativas que são posteriormente executadas.

O Android 2.2 "Froyo" trouxe a compilação *Just-in-time* (JIT) baseada em rastreamento para o Dalvik, otimizando a execução de aplicativos ao analisá-los toda vez que executassem e compilando dinamicamente segmentos curtos executados de seu *bytecode* em código de máquina nativo.

Ao contrário do Dalvik, a ART faz o uso da compilação *Ahead Of Time* (AOT) compilando aplicativos inteiros em código nativo da máquina após sua instalação. Ao eliminar a interpretação de Dalvik e a compilação JIT baseada em rastreamento, a ART melhora a eficiência geral de execução e reduz o consumo de energia, o que resulta em autonomia melhorada em dispositivos móveis. Ao mesmo tempo, a ART permite a execução mais rápida de aplicativos, contém recursos aprimorados de alocação de memória e coleta de lixo, além de novos recursos de depuração.

Para manter a compatibilidade com versões anteriores, a ART usa o mesmo *bytecode* de entrada que Dalvik, fornecido através de arquivos ".dex" (Dalvik *executable*) padrão como parte de arquivos APK, enquanto os arquivos ".odex" são substituídos por executáveis do tipo "*Executable and Linkable Format*" (ELF). Uma vez que um aplicativo é compilado no dispositivo da ART, ele é executado unicamente a partir do executável ELF compilado; Como resultado, a ART elimina várias despesas gerais de execução de aplicativos associadas à interpretação de Dalvik e compilação JIT baseada em rastreamento. Como desvantagem, a ART requer tempo adicional para a compilação quando um aplicativo é instalado, e os aplicativos ocupam quantidades ligeiramente maiores de armazenamento secundário para armazenar o código compilado (FRUMUSANU, 2014), (SOURCE ANDROID, 2017), (SOURCE ANDROID, 2017).

4.3.2. ESCALONAMENTO DE UCP

Talvez a maior dificuldade do escalonamento em sistemas operacionais seja o desejo de satisfazer simultaneamente objetivos conflitantes: tempo de resposta rápido, bom *throughput* (número de processos terminados por unidade de tempo) para processos de segundo plano, evitar postergação indefinida (*starvation*), conciliar processos de alta prioridade com os de baixa prioridade, etc. O conjunto de

regras utilizado para determinar como, quando e qual processo deverá ser executado é conhecido como política de escalonamento. Nesta subseção é abordado como o Linux/Android implementa seu escalonador e qual a política empregada para determinar quais processos recebem a UCP.

Tradicionalmente, os processos são divididos em três grandes classes: processos interativos (interação constante com o usuário), processos *batch* (sem interação com usuário), e processos tempo real (processos executados dentro de um tempo previamente determinado). Em cada classe, os processos podem ser ainda subdivididos em I/O *bound* ou *CPU bound,* de acordo com a proporção de tempo que ficam esperando por operações de entrada e saída ou utilizando a UCP. O escalonador do Android não distingue processos interativos de processos *batch*, diferenciando-os apenas dos processos de tempo real. Ele privilegia os processos I/O *bound* em relação aos *CPU bound* de forma a oferecer um melhor tempo de resposta às aplicações interativas.

O escalonador é baseado em *time-sharing*, ou seja, o tempo da UCP é dividido em fatias de tempo (*quantum*) as quais são alocadas aos processos. Se o *quantum* se esgotar durante a execução de um processo, outro processo é selecionado para execução, provocando então uma troca de contexto. Esse procedimento é completamente transparente ao processo e baseia-se em interrupções de tempo. Esse comportamento caracteriza este escalonamento como preemptivo.

O tempo de processamento é dividido em épocas (*epochs*) pelo algoritmo de escalonamento. Cada processo, no momento de sua criação, recebe um *quantum* calculado no início de uma época. O *quantum* varia de processo para processo e seu valor representa a duração de uma época. (CARISSIMI, DE OLIVEIRA e TOSCANI, 2004)

Outra característica do escalonador é a existência de prioridades dinâmicas. Ele monitora o comportamento de um processo e ajusta dinamicamente sua prioridade, visando a equalizar o uso do processador entre os processos. Processos que recentemente ocuparam o processador durante um período de tempo considerado “longo” têm sua prioridade reduzida. De forma análoga, aqueles que estão há muito tempo sem executar recebem um aumento na sua prioridade, sendo então beneficiados em novas operações de escalonamento.

Na verdade, existem dois tipos de prioridades: estática e dinâmica. As prioridades estáticas são utilizadas exclusivamente por processos de tempo real. Nesse caso, a prioridade do processo tempo real é definida pelo usuário e não é alterada pelo escalonador. Somente usuários com privilégios especiais têm a capacidade de criar e definir processos tempo real. A prioridade dinâmica é aplicada aos processos interativos e *batch*. Aqui, a prioridade é calculada levando em conta a prioridade base do processo e a quantidade de tempo restante em seu *quantum*.

Os processos de prioridade dinâmica são executados pelo escalonador apenas quando não existem processos de tempo real. Isso significa que os processos de prioridade estática tem prioridade maior que os de prioridade dinâmica.

Para selecionar um processo para execução, o escalonador prevê três políticas diferentes:

I. **SCHED_FIFO**: Essa política aplica-se apenas aos processos de tempo real. Na criação, o descritor do processo é inserido no final da fila correspondente à sua prioridade. Quando um processo é alocado ao processador, ele executa até que uma de três situações ocorra: (i) um processo de tempo real de prioridade superior torna-se apto a executar; (ii) o processo libera espontaneamente o processador para processos de prioridade igual à sua; (iii) o processo termina, ou bloqueia-se, em uma operação de entrada e saída ou de sincronização.

II. **SCHED_RR**: Na criação, o descritor do processo é inserido no final da fila correspondente à sua prioridade. Quando um processo é alocado ao processador, ele executa até que uma de quatro situações ocorra: (i) seu período de execução (*quantum*) tenha se esgotado nesse caso o processo é inserido no final de sua fila de prioridade; (ii) um processo de prioridade superior torna-se apto a executar; (iii) o processo libera espontaneamente o processador para processos de prioridade igual a sua; (iv) o processo termina, ou bloqueia-se, em uma operação de entrada e saída ou de sincronização. Essa política também só é válida para processos de tempo real.

III. **SCHED_OTHER**: Corresponde a um esquema de filas multinível de prioridades dinâmicas com *time-sharing*. Os processos interativos

pertencem a esta categoria. (CARISSIMI, DE OLIVEIRA e TOSCANI, 2004)

4.4. SISTEMA DE ARQUIVOS

Uma das características mais básicas de todas as aplicações de sistemas computacionais é a necessidade de armazenar e recuperar informações. Isso já faz do sistema de arquivos um dos tópicos centrais quando o assunto é sistemas operacionais. Não à toa já foram criados inúmeros deles como o FAT-16 (*File Allocation Table*), FAT-32, exFAT(*Extended File Allocation Table)*, NTFS (*New Technology File System*), EXT2 (*Extended File System)*, EXT3, EXT4 e outros mais.

O sistema de arquivos do Android é semelhante ao do Linux. Os arquivos podem ser agrupados em diretórios, e estes podem conter inúmeros subdiretórios, originando um sistema de arquivos hierárquico. O caractere "/" é o diretório raiz e obviamente contem subdiretórios. O caractere "/" também é utilizado para separar nomes de diretórios, de modo que */dir1/dir2/arq* representa um arquivo localizado no diretório *dir2* que por sua vez está contido no diretório *dir1.*

À um usuário regular, isto é, sem permissões de acesso administrativas (*root)*, o Android exibe apenas os diretórios aos quais o usuário tem acesso. Dessa forma, apenas uma parte de toda a hierarquia de arquivos é exibida ao usuário.

No Android, todos os arquivos e diretórios de uma aplicação fluem através de uma camada de abstração do *kernel* chamado *Virtual File System* (VFS)[3]. Cada sistema de arquivos, e há muitos deles, são implementações do VFS. Cada sistema de arquivos possui um módulo de *kernel* separado que registra as operações que ele efetua com o VFS.

Ao separar a implementação da abstração, a adição de um novo sistema de arquivos torna-se uma questão de escrever outro módulo do *kernel*. Esses módulos

[3] O VFS (*Virtual File System*) é o mecanismo que permite que chamadas de sistemas genéricas, tais como *open*() e *read*(),possam ser executadas independentemente do sistema de arquivos usado ou do meio físico.

são parte do *kernel*, ou são carregados dinamicamente sob demanda. O *kernel* do Android vem com um subconjunto da extensa coleção de sistemas de arquivos. Toda função de gerenciamento é transparente ao usuário, pois o *kernel* lida com todo o trabalho quando ele monta um sistema de arquivos.

Como existem inúmeros fabricantes de dispositivos e de aparelhos Android em si, os sistemas de arquivos suportados variam de aparelho para aparelho, mas os sistemas de arquivos comuns de memória *flash* são:

- **ExFAT:** É um sistema de arquivos de propriedade da Microsoft para memórias flash e é empregado por parte dos fabricantes de dispositivos.
- **F2FS:** O *Flash-Friendly File System* foi apresentado pela Samsung em 2012 como um sistema de arquivos *open source* do Linux.
- **JFFS2:** Chamado de *Journal Flash File System* (2º versão) é o sistema de arquivos padrão para muitos fabricantes de dispositivos Android desde a versão 4.0 *Ice Cream Sandwich*.

4.5. GERENCIAMENTO DE MEMÓRIA

A memória principal é um recurso extremamente importante para qualquer tipo de sistema computacional, que deve ser cuidadosamente gerenciado. Graças a diminuição do valor comercial deste tipo de memória, os computadores atuais as possuem em quantidade proporcionalmente muito maior que os sistemas de décadas atrás, porém, a exigência de memória por parte dos programas aumentou mais rapidamente que o incremento desse recurso. Dessa forma, a memória continua sendo um recurso crítico e, caso não seja gerenciada adequadamente, pode comprometer consideravelmente o desempenho do sistema.

Parte das funções do sistema operacional é garantir que cada processo tenha seu próprio espaço na memória, além de prover a proteção deste espaço para impedir que seja sobrescrito por outros processos. Também é preciso assegurar que uma aplicação não tente utilizar mais memória do que a existente no sistema.

Como mencionado anteriormente, o Android utiliza o *kernel* Linux, de modo que este é o responsável pelo gerenciamento da memória. Na verdade, todas as operações básicas dos níveis mais baixos do sistema operacional, como o

gerenciamento de processos e energia, o controle de operações de entrada/saída, e outros ficam a cargo do *kernel* Linux.

4.5.1. PRIORIDADE E STATUS DE PROCESSOS

O Android mantém todos os processos na memória até que surja a necessidade de alocar recursos para outros processos. Quando a quantidade de memória principal disponível é considerada baixa, é utilizado o “Matador de Falta de Memória” (*Low Memory Killer)*, que é uma extensão do Linux, assim como os *Wake Locks*, que serão discutidos na seção 4.7, sua função é encerrar processos que tenham o menor impacto possível para o que quer que o usuário esteja fazendo, de modo que a quantidade de memória principal disponível passe a ser suficiente para que o sistema jamais entre em estados de paginação ruins, que podem impactar negativamente na experiência do usuário. No entanto, uma questão chave se faz presente: Qual processo deve ser encerrado?

Torna-se necessário estabelecer uma ordem para definir quais serão os primeiros processos a serem finalizados em caso de necessidade. Tal ordem está associada ao nível de prioridade de cada processo. A figura 20 ilustra os possíveis estados e níveis de prioridade de cada processo contido na memória.

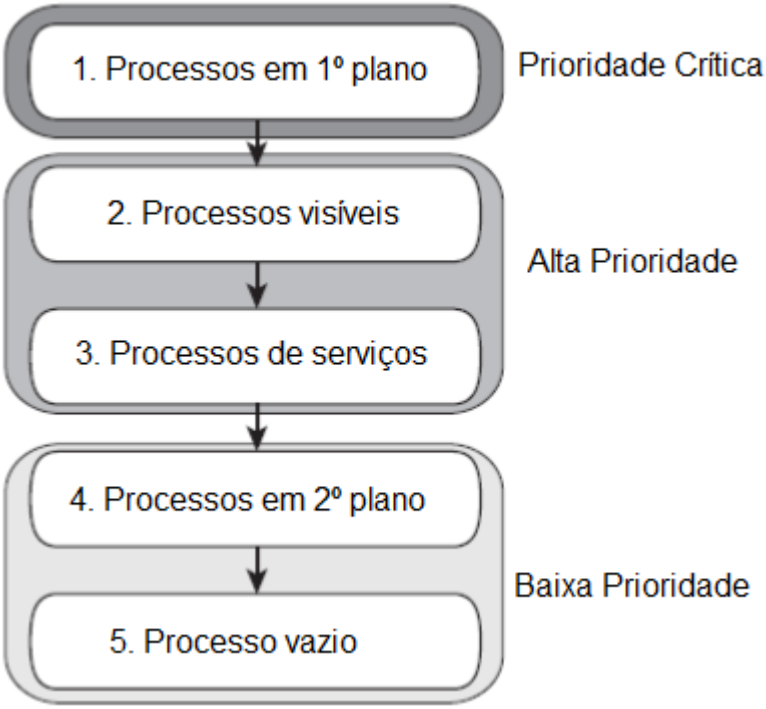

Figura 20. *Estados e Prioridades dos Processos.*

Fonte: (MOBWORLD, 2010).

- **Processos em 1º plano:** Os processos ativos são aqueles que hospedam aplicações cujos componentes estão atualmente em interação com o usuário. Normalmente essa categoria engloba poucos processos, e eles serão eliminados apenas como último recurso.
- **Processos Visíveis*:*** Processos sem componentes em primeiro plano, mas que ainda possam afetar o que o usuário vê na tela. Um processo visível é considerado extremamente importante e não será eliminado a não ser que seja necessário para manter em execução os processos em primeiro plano.
- **Processos de serviços:** Processos que executem um serviço que tenha sido iniciado. Embora os processos de serviço não estejam diretamente ligados a tudo o que os usuários veem, normalmente eles desempenham tarefas as quais o usuário se importa (como reproduzindo música em segundo plano ou baixando dados na rede) e o sistema os mantém em execução a menos que não haja memória suficiente para mantê-los juntamente com todos os processos visíveis e em primeiro plano.

- **Processos de 2º plano:** Processos que realizam uma atividade que não está diretamente visível para o usuário. Esses processos não têm impacto direto na experiência do usuário e o sistema poderá eliminá-los a qualquer momento para liberar memória para processos em primeiro plano, visíveis ou de serviço. Geralmente há vários processos em segundo plano em execução e eles são mantidos em uma lista com escalonamento LRU (*Least Recently Used*) para garantir que o processo cuja atividade foi exibida por último pelo usuário seja o último a ser eliminado.
- **Processo Vazio:** Um processo que não tem nenhum componente de aplicativo ativo. O único motivo para manter esse tipo de processo ativo é para armazenamento em cache, a fim de reduzir seu tempo de inicialização na próxima vez em que ele for executado.

O Android coloca cada processo no maior nível possível, com base na importância dos componentes atualmente ativos no processo. Por exemplo, se um processo hospedar um serviço e uma atividade visível, ele terá a classificação de um processo visível, e não a de um processo de serviço. Além disso, a classificação de um processo pode ser elevada porque outros processos dependem dele — um processo que está servindo a outro processo nunca pode ter classificação menor do que a do processo que está sendo servido.

Se dois processos possuírem o mesmo nível de prioridade, como critério de desempate o processo que teve histórico de prioridade mais baixa durante mais tempo é escolhido para ser finalizado primeiro (ANDROID, 2017).

4.6. SEGURANÇA E ABERTURA

Considerando que atualmente um grande número de aplicações está disponível e quase sempre ao instalar uma delas não sabemos o quão confiável seu projetista é, a segurança passa a ser um aspecto crítico do sistema.

Segundo Tanenbaum & Bos (2016), para o Android, uma aplicação é como se fosse um hóspede (ou convidado) executando no dispositivo do usuário. Assim, a

uma aplicação nada é confiado, a menos que o usuário permita explicitamente. Na implementação do Android, essa ideia é expressa através dos IDs dos usuários. Quando uma aplicação é instalada, um novo ID de usuário único do Linux (UID – *User Identification*) é criado para ela, e todo seu código executa como aquele "usuário". Dessa maneira é criada uma 'caixa de areia' (*sandbox)* para cada aplicação, com sua própria área isolada do sistema de arquivos. Através dessa política, o Android utiliza uma característica já existente no Linux de uma maneira nova. O resultado é um melhor isolamento entre diferentes aplicações e o sistema.

Uma série de UIDs padrão é definida pelo Android para as partes de nível inferior do sistema, no entanto, a maior parte das aplicações tem seu UID designado dinamicamente, em sua primeira inicialização ou no momento da instalação.

Os UIDs que podem ser designados às aplicações pelo gerenciador de pacotes estão compreendidos na faixa de 10.000 a 19.999; dessa forma, o número máximo de aplicações instaladas no sistema é 10.000.

Assim que uma aplicação recebe seu UID, um novo diretório de armazenamento é criado para ela. A aplicação recebe livre acesso para seus arquivos privados ali, mas não tem permissão para acessar os arquivos de outras aplicações, assim como estas não podem acessar os arquivos da nova aplicação. Isso faz com que os provedores de conteúdo (vistos na seção 4.2.4) tenham grande importância, uma vez que são uns dos poucos mecanismos que podem transferir dados entre aplicações.

Até o próprio sistema, que executa com UID 1000, não pode acessar os arquivos das aplicações. E é por conta disso que o *daemon installd* existe: ele executa com privilégios especiais para ser capaz de acessar e criar diretórios e arquivos para outras aplicações. O *installd* fornece uma API muito restrita ao gerenciador de pacotes para que ele crie e gerencie diretórios de dados de aplicações conforme necessário.

Sob o estado original, as 'caixas de areia' do Android não devem permitir quaisquer interações entre aplicações que possam violar a segurança entre elas. Isso acontece não só para conferir robustez (impedir que um aplicativo "quebre" outro) ao sistema, mas também diz respeito a confidencialidade das informações.

Como exemplo, considere o aplicativo da câmera. Quando o usuário tira uma foto, ela é armazenada pelo aplicativo em seu espaço de dados privado.

Nenhuma outra aplicação tem acesso àqueles dados, que é o desejável uma vez que fotos podem representar dados sensíveis para o usuário.

Considere que após o usuário ter tirado essa foto, ele deseje enviá-la por *e-mail* para um amigo. O *e-mail* é um aplicativo separado, com sua própria 'caixa de areia' e, consequentemente, sem acesso a qualquer foto na aplicação da câmera. É preciso fazer com que o aplicativo de *e-mail* consiga acesso às fotos da câmera em sua própria 'caixa de areia'.

A forma mais conhecida de controle de acesso no Android são as permissões de aplicação. Permissões são capacidades específicas e bem definidas, que podem ser concedidas a um aplicativo no momento da instalação. O aplicativo lista as permissões que ele precisa em seu manifesto, e antes de instalar o aplicativo, o usuário é informado do que será permitido fazer baseado nelas.

A Figura 21 ilustra como o aplicativo de *e-mail* pode utilizar as permissões para acessar as fotos no aplicativo da câmera. Nesse caso, o aplicativo da câmera tem associado consigo a permissão *READ_PICTURES* com suas fotos, informando que qualquer aplicativo com posse dessa permissão pode acessar seus dados de fotos. A aplicação de *e-mail* agora pode acessar o URI de propriedade da câmera como *content://pics/1*; o provedor de conteúdo do aplicativo de câmera, ao receber a solicitação para sua URI, questiona o gerenciador de pacotes se o aplicativo solicitante tem a permissão necessária. Em caso afirmativo a chamada é bem-sucedida e os dados são retornados à aplicação.

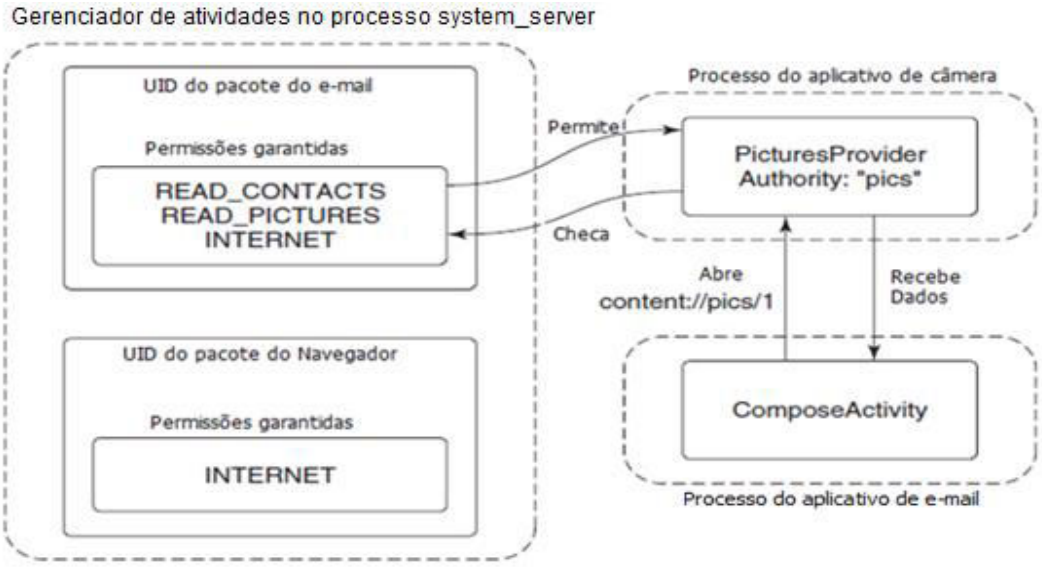

Figura 21. *Solicitando e usando uma permissão.*

Fonte: (TANENBAUM e BOS, 2016).

A Figura 22 mostra o que acontece quando uma aplicação não tem a permissão necessária para uma operação que ele está efetuando. Nesse caso, o aplicativo do navegador tenta acessar as fotos do usuário diretamente, mas ele tem só a permissão para operações em rede pela internet. Dessa forma, o *PicturesProvider* recebe a informação do gerenciador de pacotes que o processo solicitante não tem a permissão *READ_PICTURES* que é necessária, e por conta disso uma *SecurityException* é lançada de volta para o processo solicitante.

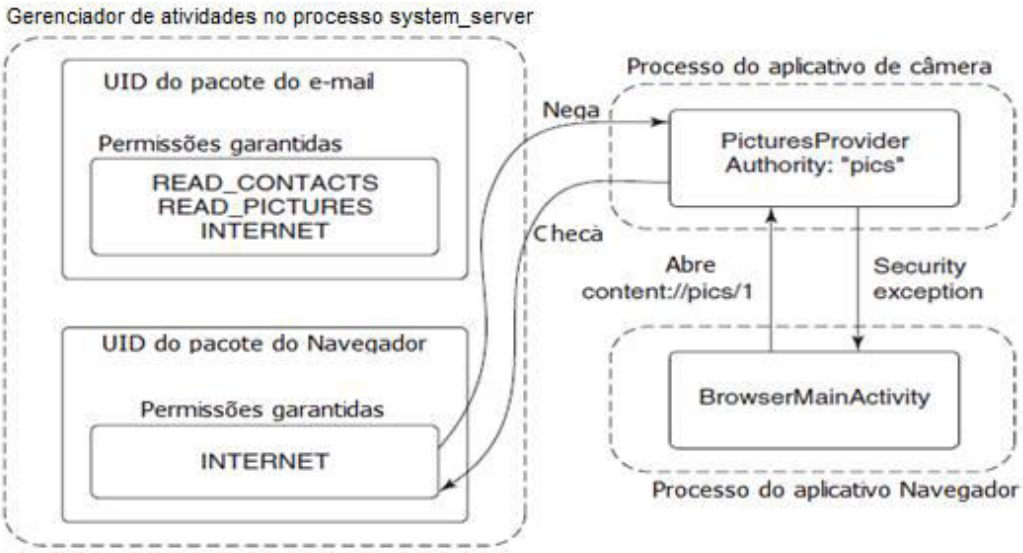

Figura 22. *Acessando dados sem permissão.*
Fonte: (TANENBAUM e BOS, 2016).

O ecossistema móvel abrange uma série de "personagens". Entre eles estão os manufatureiros, que produzem e vendem dispositivos móveis; fornecedores de serviços (operadores), que oferecem conexão de rede e outros serviços que agregam valor; consumidores; desenvolvedores que projetam aplicações e contribuem com o desenvolvimento e evolução do sistema operacional, caso este seja *open source*.

Todos estes "personagens" são afetados pela “abertura” do sistema operacional. Para os operadores, esta característica determinará o quão fácil seus serviços poderão ser portados, migrados, implantados e executados em diferentes dispositivos. Para os manufatureiros, a “abertura” define o quão personalizável o sistema operacional será para funcionar em diferentes plataformas e diferenciar seus dispositivos uns dos outros. Para os desenvolvedores, ela determina o quão fácil eles conseguirão desenvolver novas aplicações e como seus investimentos nesse setor serão maximizados pelo fato de sua aplicação, escrita uma vez, poder ser executada em diferentes dispositivos. Para os usuários finais, define qual será a facilidade em conseguir novas aplicações sem se preocupar muito com deficiências de interface e experiência de usuário, além de uma possível incompatibilidade com seu dispositivo (LI, WANG, *et al.*, 2012).

Um aspecto delicado, no entanto, que não pode deixar de ser avaliado é a dualidade segurança/abertura. Enquanto a primeira não deseja expor todas as funcionalidades do sistema para entidades externas, uma vez que isso coloca o sistema sob ameaça de segurança, a segunda deseja ser explorada em sua plenitude, de forma a beneficiar os "personagens" descritos anteriormente.

Assim, este ponto é crucial, uma vez que tem um peso considerável no apelo que o sistema operacional terá frente a todos os grupos envolvidos neste ecossistema. Pode-se dizer que parte do grande sucesso do Android é devido a um bom equilíbrio entre a segurança e sua abertura.

4.7. GERENCIAMENTO DE ENERGIA

O gerenciamento de energia é, e continuará sendo uma questão chave para os projetistas de sistemas operacionais móveis. A demanda por energia nos dispositivos móveis é crescente, uma vez que as aplicações desenvolvidas para essa plataforma são cada vez mais poderosas e, portanto, exigem cada vez mais recursos. As baterias, por sua vez, não conseguem apresentar o mesmo ritmo de melhorias em relação às aplicações, parte disso tem origem no lento desenvolvimento de sua tecnologia e a preferência das pessoas por dispositivos pequenos e que caibam no bolso. (LI, WANG, *et al.*, 2012)

4.7.1. GERENCIAMENTO DE ENERGIA NO PROCESSADOR

Sistemas Operacionais móveis têm feito um progresso estável em relação ao gerenciamento de energia na última década. Inicialmente o foco era na gerência de energia do processador, pois tratava-se do componente que realizava o consumo mais significativo de energia. Processadores modernos dão suporte a frequência dinâmica e a escala de voltagem, o que permite ao sistema operacional ajustar a frequência e a voltagem dinamicamente do processador em tempo de execução baseado na demanda de energia computacional exigida pela carga de trabalho que está atualmente no processador, o que economiza uma quantidade considerável de energia. Em adição a frequência dinâmica e a escala de voltagem, processadores

modernos tipicamente suportam múltiplos processos em estado de inatividade com diferentes quantidades de energia consumida nesses estados inativos. Quanto mais profundo o estado inativo, mais energia pode ser economizada (LI, WANG, *et al.*, 2012).

4.7.2. GERENCIAMENTO DE ENERGIA NO DISPOSITIVO

Dentre as várias funcionalidades dos sistemas operacionais para dispositivos móveis, uma das mais relevantes atualmente é o gerenciamento de energia. Em particular, um mecanismo foi introduzido para gerenciar a energia dos dispositivos de entrada e saída em tempo de execução, colocando automaticamente dispositivos de E/S em qualquer estado de energia baixa apropriada que eles suportem quando os dispositivos correspondentes são detectados como ociosos em tempo de execução.

Além do gerenciamento de energia dos dispositivos de E/S enquanto estão inativos, existem algumas inovações tecnológicas para poupar a energia de tais dispositivos enquanto eles estão ativos, por exemplo, *GPUs* (*Graphics Processing Unit)* modernas estão começando a suportar a tensão dinâmica e escala de frequência semelhante à encontrada nos processadores. Tais recursos podem reduzir o consumo de energia em até 50% para gráficos 3D em alguns casos. Além disso, dispositivos de entrada e saída estão se tornando mais inteligentes em relação ao fato de poderem trabalhar isoladamente, sem a intervenção do processador (LI, WANG, *et al.*, 2012).

4.7.3. *WAKE LOCKS* (TRAVAS DE DESPERTAR)

Considerando que os dispositivos móveis possuem particularidades em relação ao gerenciamento de energia quando comparados aos sistemas tradicionais de computação, foi necessária a criação de um componente com a incumbência de gerir a energia. Os *Wake Locks*, que assim como o Matador de Falta de Memória

são algumas das inovações do Android em relação ao Linux, decidem quando o sistema vai “dormir”.

Segundo Tanenbaum & Bos (2016), em um sistema computacional tradicional, o sistema pode estar em um de dois estados de energia: executando e pronto para a entrada do usuário, ou profundamente adormecido. Enquanto executa, *hardwares* secundários podem ser ligados ou desligados conforme a necessidade, mas a UCP em si e partes centrais do *hardware* têm de permanecer ativas para lidar com o tráfego de rede e outros eventos dessa natureza. Ir para o estado de sono mais profundo é algo que acontece de maneira relativamente rara: seja através do usuário colocando explicitamente o sistema para dormir, ou o sistema indo dormir por causa de um intervalo um tanto longo de inatividade do usuário. Sair do estado de sono exige uma interrupção de *hardware* de uma fonte externa, como pressionar um botão ou um teclado, ponto no qual o dispositivo vai despertar e ligar sua tela.

Naturalmente, usuários de *smartphones* têm expectativas diferentes. Embora seja comum desligar a tela do dispositivo como se tivesse colocado o *smartphone* para dormir, o usuário não deseja que o sistema entre em um estado de sono profundo. Mesmo com a tela desligada o dispositivo precisa continuar ativo para conseguir realizar algumas tarefas como receber telefonemas, receber e processar dados de mensagens, exibir notificações (via vibração e som, por exemplo), dentre outras.

Dito isso, uma solução simples seria não permitir que a UCP durma quando a tela do dispositivo estiver desligada, afinal, o *kernel* sabe quando não há trabalho programado para quaisquer *threads*, e o Linux automaticamente deixará a UCP ociosa e usará menos energia nessa situação. No entanto, uma UCP ociosa não é a mesma coisa que o sono de verdade. Em muitos conjuntos de *chips*, o estado ocioso consome significativamente mais energia do que em um estado de sono verdadeiro.

Travas de despertar no Android permitem que o sistema entre em um modo de sono mais profundo, sem estar vinculado a nenhuma ação explícita do usuário como desligar a tela. O estado padrão do sistema com travas de despertar é que o dispositivo está dormindo. Quando ele está executando, a fim de evitar que ele durma, é preciso que (pelo menos) uma trava de despertar esteja sendo mantida.

Enquanto a tela estiver ligada, obviamente o sistema sempre mantém uma trava de despertar, evitando que o dispositivo vá dormir, e então ele seguirá executando. Quando nenhuma trava de despertar for mantida, o sistema vai dormir, e ele sairá do sono somente por uma interrupção de *hardware*. Algumas dessas interrupções podem ser alarmes baseados em tempo, uma chamada telefônica que chega ou botões sendo pressionados.

5. CONSIDERAÇÕES FINAIS

Através da análise do abrangente referencial técnico e teórico coletado durante a elaboração do presente estudo, pode-se afirmar que os dispositivos e sistemas operacionais móveis estão se tornando o principal sistema computacional da maioria dos usuários e tendem a se estabelecer cada vez mais. A computação móvel é um dos segmentos mais importantes da computação atualmente, englobando subáreas como a Internet das Coisas, a Computação Ubíqua e Pervasiva entre outras. Considerando que estes segmentos apresentam grande potencial tecnológico e financeiro, uma vez que ainda estão longe de serem exploradas por completo, certamente figurarão entre as áreas de maior destaque por um longo tempo.

Através deste estudo foi possível consolidar um conhecimento abrangente sobre o "ecossistema" móvel, partindo de tópicos com abordagens informativas como a distribuição de usuários nos últimos anos, assistentes pessoais virtuais, desenvolvimento de aplicações, sistemas operacionais móveis mais antigos até a abordagem mais profunda e específica que foi realizada sobre o sistema operacional Android.

É bem provável que o Android mantenha essa popularidade durante um bom tempo. O sistema já está muito consolidado em uma grande variedade de dispositivos (não só em *smartphones* e *tablets* como já fora mencionado) e dispõe de uma grande quantidade de entusiastas e programadores dispostos a adaptá-lo sempre que necessário para utilizá-lo em um novo ambiente.

Apesar de a computação estar em constante transformação, o que periodicamente acaba criando novas subáreas, projeções e perspectivas, uma grande inovação, como a apresentada nesta pesquisa, normalmente só ocorre quando existe uma base muito bem consolidada, de modo a impulsioná-la e torná-la consistente. E essa é a relação entre os sistemas operacionais "convencionais" e os móveis.

BIBLIOGRAFIA

ALGAZE, B. Microsoft saw the future, but missed creating it. **ExtremeTech**, 2015. Disponivel em: <https://www.extremetech.com/computing/206519-microsoft-saw-the-future-but-missed-creating-it>. Acesso em: 13 Dez. 2016.

ALVES, P. A evolução do Android: do Cupcake ao Marshmallow, conheça todas as versões. **TechTudo**, 2015. Disponivel em: <http://www.techtudo.com.br/noticias/noticia/2015/10/a-evolucao-do-android-do-cupcake-ao-marshmallow-conheca-todas-as-versoes.html>. Acesso em: 30 Ago. 2017.

ANDROID. Arquitetura da plataforma. **Android**, 2017. Disponivel em: <https://developer.android.com/guide/platform/index.html>. Acesso em: 31 Out. 2017.

ANDROID. Processos e Encadeamentos. **Android**, 2017. Disponivel em: <https://developer.android.com/guide/components/processes-and-threads.html>. Acesso em: 25 Out. 2017.

AQUINO, J. F. S. **Plataformas de Desenvolvimento para Dispositivos Móveis**. PUC-RJ. Rio de Janeiro. 2007.

BORDIN, M. V. **Introdução a Arquitetura Android**. Sociedade Educacional Três de Maio. Três de Maio. 2012.

CARISSIMI, A. D. S.; DE OLIVEIRA, R. S.; TOSCANI, S. S. **Sistemas Operacionais**. 3º. ed. Porto Alegre: Bookman, 2004.

CARRARA, D. Google barra interface das fabricantes no Android Wear. **AndroidPit**, 2014. Disponivel em: <https://www.androidpit.com.br/google-impede-interface-fabricantes-android-wear>. Acesso em: 3 Out. 2017.

CHINADAILY. Top 10 highs and lows for Nokia in China. **Chinadaily**, 2014. Disponivel em: <http://usa.chinadaily.com.cn/business/2014-10/24/content_18797233_4.htm>. Acesso em: 15 Dez. 2016.

CIRIACO, D. Apple apresenta oficialmente o novo iOS 11; confira todas as novidades. **TecMundo**, 2017. Disponivel em: <https://www.tecmundo.com.br/ios-11/117413-apple-apresenta-oficialmente-novo-ios-11.htm>. Acesso em: 29 Ago. 2017.

CORDEIRO, F. Android SDK: O que é? Para que Serve? Como Usar? **Android Pro**, 2017. Disponivel em: <http://www.androidpro.com.br/android-sdk/>. Acesso em: 2 Out. 2017.

CUNNINGHAM, A. What happened to the Android Update Alliance? **arstechnica**, 2012. Disponivel em: <https://arstechnica.com/gadgets/2012/06/what-happened-to-the-android-update-alliance/>. Acesso em: 30 Set. 2017.

DAVIES, C. Windows Phone “Mango” official; Acer, Fujitsu and ZTE onboard. **Slash Gear**, 2011. Disponivel em: <https://www.slashgear.com/windows-phone-mango-official-acer-fujitsu-and-zte-onboard-24153926/>. Acesso em: 30 Ago. 2017.

FRUMUSANU, A. A Closer look at Android RunTime (ART) in Android L. **AnandTech**, 2014. Disponivel em: <https://www.anandtech.com/show/8231/a-closer-look-at-android-runtime-art-in-android-l>. Acesso em: 27 Nov. 2017.

G1. Android passa Windows e se torna o sistema operacional mais usado do mundo. **G1**, 2017. Disponivel em: <https://g1.globo.com/tecnologia/noticia/android-passa-windows-e-se-torna-o-sistema-operacional-mais-usado-do-mundo.ghtml>. Acesso em: 11 Set. 2017.

G1. Microsoft encerra suporte oficial ao Windows Phone 8.1. **G1**, 2017. Disponivel em: <https://g1.globo.com/tecnologia/noticia/microsoft-encerra-suporte-oficial-ao-windows-phone-81.ghtml>. Acesso em: 30 Ago. 2017.

GOMES, R. C.; FERNANDES, J. A. R.; FERREIRA, V. C. **Sistema Operacional Android**. UFF. Niterói. 2012.

HIGA, P. 7 melhores novidades do Android 7.0 Nougat. **Tecnoblog**, 2016. Disponivel em: <https://tecnoblog.net/198297/android-7-nougat-novidades/>. Acesso em: 30 Ago. 2017.

HOLLISTER, S. Sony Smart Watch (aka Sony Ericsson LiveView 2) hands-on. **TheVerge**, 2012. Disponivel em: <https://www.theverge.com/2012/1/10/2695959/sony-smart-watch-aka-sony-ericsson-liveview-2-hands-on>. Acesso em: 30 Ago. 2017.

KLEINA, N. A história do Android, o robô que domina o mercado mobile. **TecMundo**, 2017. Disponivel em: <https://www.tecmundo.com.br/ciencia/120933-historia-android-robo-domina-o-mercado-mobile-video.htm>. Acesso em: 29 Ago. 2017.

LI, X.-F. et al. Mobile OS Architecture Trends. **Intel Technology Journal**, p. 5, 2012.

LIEN, T. Ouya Kickstarter campaign ends with more than $8.5 million raised. **TheVerge**, 2012. Disponivel em: <https://www.theverge.com/2012/8/9/3229673/ouya-kickstarter-campaign-ends-with-8580359-raised>. Acesso em: 03 Out. 2017.

MCLOUGHLIN, J. S. Sync Palm OS Handhelds With Mac OS X Snow Leopard Using The Missing Sync for Palm OS. **PalmAddicts**, 2009. Disponivel em: <http://palmaddict.typepad.com/palmaddicts/2009/08/sync-palm-os-handhelds-with-mac-os-x-snow-leopard-using-the-missing-sync-for-palm-os.html>. Acesso em: 12 Dez. 2016.

MEYER, M. A História do iOS. **Oficina da Net**, 2016. Disponivel em: <https://www.oficinadanet.com.br/post/17950-a-historia-do-ios>. Acesso em: 29 Ago. 2017.

MOBWORLD. Memory Management in Android. **mobworld**, 2010. Disponivel em: <https://mobworld.wordpress.com/2010/07/05/memory-management-in-android/>. Acesso em: 22 Set. 2017.

MORIMOTO, C. E. Smartphones: a história do Symbian. **Hardware**, 2010. Disponivel em: <http://www.hardware.com.br/dicas/historia-symbian.html>. Acesso em: 19 Set. 2017.

MULLER, L. iOS 10: tudo sobre o novo SO mobile da Apple. **TecMundo**, 2016. Disponivel em: <https://www.tecmundo.com.br/wwdc-2016/106038-ios-10-tudo-novo-so-mobile-apple.htm>. Acesso em: 29 Ago. 2017.

MULLER, L. Android Oreo: conheça 10 novidades do novo SO da Google. **TecMundo**, 2017. Disponivel em: <https://www.tecmundo.com.br/software/121059-android-oreo-conheca-10-novidades-novo-so-google.htm>. Acesso em: 30 Ago. 2017.

PALM OS. Palm OS | Palm Source. **Palm Source**, 2010. Disponivel em: <https://www.palmsource.com/palmos/>. Acesso em: 28 Set. 2016.

PESTRE, M. C. iOS 9 tem economia de bateria, Siri proativa e aplicativos novos. **Techtudo**, 2015. Disponivel em: <http://www.techtudo.com.br/tudo-sobre/ios-9.html>. Acesso em: 29 Ago. 2017.

PETROVAN, B. Android Everywhere: 10 Types of Devices That Android Is Making Better. **Android Authority**, 2012. Disponivel em: <https://www.androidauthority.com/android-everywhere-10-types-of-devices-that-android-is-making-better-57012/>. Acesso em: 30 Ago. 2017.

PUREWAL, S. J. The difference between Google Now and Google Assistant. **cnet**, 2016. Disponivel em: <https://www.cnet.com/how-to/the-difference-between-google-now-and-google-assistant/>. Acesso em: 2 Out. 2017.

REUTERS. Android supera Symbian e assume liderança em smartphones. **g1**, 2011. Disponivel em: <http://g1.globo.com/tecnologia/noticia/2011/01/android-assume-lideranca-em-smartphones-diz-canalys.html>. Acesso em: 30 Ago. 2017.

RINALDI, C. Android Wear: tudo sobre o SO para wearables da Google. **AndroidPit**, 2015. Disponivel em: <https://www.androidpit.com.br/android-wear-novidades-recursos-dispositivos>. Acesso em: 3 Out. 2017.

RUBINO, D. Overview and review of Windows Phone 8. **Windows Central**, 2012. Disponivel em: <https://www.windowscentral.com/overview-and-review-windows-phone-8>. Acesso em: 30 Ago. 2017.

SANTINO, R. Android já teve 10 versões diferentes; relembre a evolução do sistema. **Olhar Digital**, 2013. Disponivel em: <https://olhardigital.com.br/noticia/android-ja-teve-10-versoes-diferentes-relembre-a-evolucao-do-sistema/35801>. Acesso em: 30 Ago. 2017.

SANTINO, R. Menos de 1% do mercado: os 5 motivos que levaram o Windows Phone ao buraco. **Olhar Digital**, 2016. Disponivel em:

<https://olhardigital.com.br/noticia/menos-de-1-do-mercado-os-5-motivos-que-levaram-o-windows-phone-ao-buraco/58615>. Acesso em: 30 Ago. 2017.

SILVA, R. Samsung Galaxy Camera é uma câmera compacta que roda Android. **Tecnoblog**, 2012. Disponivel em: <https://tecnoblog.net/122229/galaxy-camera-hands-on/>. Acesso em: 30 Ago. 2017.

SOURCE ANDROID. ART and Dalvik. **Android**, 2017. Disponivel em: <https://source.android.com/devices/tech/dalvik/>. Acesso em: 27 Nov. 2017.

SOURCE ANDROID. Configuring ART. **Source Android**, 2017. Disponivel em: <https://source.android.com/devices/tech/dalvik/configure#how_art_works>. Acesso em: 27 Nov. 2017.

STATCOUNTER. Mobile Operating System Market Share Worldwide. **StatCounter**, 2012. Disponivel em: <http://gs.statcounter.com/os-market-share/mobile/worldwide#monthly-201201-201212>. Acesso em: 20 Ago. 2017.

STATCOUNTER. Mobile Operating System Market Share Worldwide. **StatCounter**, 2013. Disponivel em: <http://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201301-201312>. Acesso em: 20 Ago. 2017.

STATCOUNTER. Mobile Operating System Market Share Worldwide. **StatCounter**, 2014. Disponivel em: <http://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201401-201412>. Acesso em: 20 Ago. 2017.

STATCOUNTER. Mobile Operating System Market Share Worldwide. **StatCounter**, 2015. Disponivel em: <http://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201501-201512>. Acesso em: 20 Ago. 2017.

STATCOUNTER. Frequently Asked Questions. **Global Stats StatCounter**, 2017. Disponivel em: <http://gs.statcounter.com/faq#methodology>. Acesso em: 18 Setembro 2017.

STATCOUNTER. Mobile Operating System Market Share Brazil. **StatCounter**, 2017. Disponivel em: <http://gs.statcounter.com/os-market-share/mobile/brazil/#monthly-201707-201708>. Acesso em: 20 Ago. 2017.

STATCOUNTER. Mobile Operating System Market Share United States Of America. **StatCounter**, 2017. Disponivel em: <http://gs.statcounter.com/os-market-share/mobile/united-states-of-america/#monthly-201707-201708>. Acesso em: 20 Ago. 2017.

STATCOUNTER. Operating System Market Share Worldwide. **StatCounter**, 2017. Disponivel em: <http://gs.statcounter.com/os-market-share#monthly-201708-201708-bar>. Acesso em: 20 Ago. 2017.

STATCOUNTER. Mobile Operating System Market Share Worldwide. **StatCounter**, 2017. Disponivel em: <http://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201608-201708>. Acesso em: 20 Ago. 2017.

TANENBAUM, Andrew S; BOS, Herbert. **Sistemas Operacionais Modernos**. Traduzido por: Jorge Ritter. Revisão Técnica: Raphel Y. de Camargo. 4. Ed. São Paulo: Pearson Education do Brasil, 2016.

VINCENT, J. 99.6 percent of new smartphones run Android or iOS. **TheVerge**, 2017. Disponivel em: <https://www.theverge.com/2017/2/16/14634656/android-ios-market-share-blackberry-2016>. Acesso em: 16 Set. 2017.

WEBER, T. Nokia celebrates 1.5bn phones running on S40. **BBC**, 25 Janeiro 2012. Disponivel em: <http://www.bbc.com/news/business-16712308>. Acesso em: 28 Ago. 2017.

YADAV, A.; MAHESHWARI, S. A Comparative Scrutiny of Smartphone Operating Systems. **International Journal Series**, Bhopal, v. II, 2016.

ZAP, A. iOS: História e Evolução. **Programadores do Futuro**, 2015. Disponivel em: <https://programadoresdofuturo.wordpress.com/2015/09/14/ios-historia-e-evolucao/>. Acesso em: 29 Ago. 2017.