MEDIOtec

Curso Técnico em Informática

Arquitetura de Sistemas *Mobile*

Ronald Emerson Scherolt da Costa

Presidência da República Federativa do Brasil
Ministro da Educação
Secretaria de Educação Profissional e Tecnológica
Instituto Federal de Brasília – IFB

Reitora
Luciana Miyoko Massukado

Pró-Reitora de Ensino – PREN
Yvonete Bazbuz da Silva Santos

Diretora de Educação a Distância – DEaD
Jennifer de Carvalho Medeiros

Apoio Administrativo
Cláudia Sabino Fernandes
Joscélia Moreira de Azevedo
Noeme César Gonçalves

Coordenador Geral - DEaD
Hênio Delfino Ferreira de Oliveira

Coordenadora Adjunta Administrativo
Cláudia Sabino Fernandes

Coordenadora Adjunta Ensino
Luciana Brandão Dourado

Coordenadora Adjunta Produção de Conteúdos para EaD
Sylvana Karla da Silva de L. Santos

Coordenador Adjunto Tecnologia
Hugo Silva Faria

Orientador de Ensino e Aprendizagem
Jefferson Amauri Leite de Oliveira

Coordenador Curso Técnico em Informática
Heitor Barros

Equipe de Elaboração

Conteudista
Ronald Emerson Scherolt da Costa

Revisora de Conteúdo
Elaine Menezes

Ilustrador e diagramador
Márlon Cavalcanti Lima

Gestora de Ambiente Virtual de Aprendizagem
Daniele Cadête de Araujo Lima

Catalogação na fonte pela Biblioteca do instituto Federal de Educação, Ciência e Tecnologia - Brasília

Sumário

Sumário

Olá! Tudo bem?

O tema **"Arquitetura de sistemas *mobile*"** busca despertar o interesse e a capacidade em conhecer a arquitetura e os conceitos de sistemas operacionais (SO) para *Desktop* e dispositivos *Mobile*, assim como articular os conhecimentos a respeito do funcionamento dos seus módulos de gerência.

Compreender o correto funcionamento dos módulos de gerência de um sistema operacional irão contribuir para melhorar a gestão dessas plataformas, como também, permitirão melhorar o desenvolvimento de sistemas e aplicativos para esses dispositivos.

A estrutura e o funcionamento de um SO são tópicos bastante abrangentes e extremamente importantes. Um SO não é executado como uma aplicação sequencial, com início, meio e fim. Suas rotinas são executadas sem uma ordem predefinida e compreender esse processo é importante tanto para um técnico, um administrador de sistemas ou para um desenvolvedor.

Este livro está estruturado em 4 unidades temáticas com tópicos específicos.

Na Unidade 01, será abordada a introdução aos sistemas operacionais e sua história.

Na Unidade 02, serão tratados os conceitos básicos que envolvem o funcionamento dos sistemas operacionais, sua gerência interna e as suas funcionalidades.

Na Unidade 03, você irá aprender a articular os conhecimentos de sistemas operacionais, relacionando-os com os sistemas operacionais para dispositivos móveis.

Por fim, na Unidade 04, você iniciará uma jornada para conhecer o conceito de máquinas virtuais e a sua aplicação para dispositivos móveis.

Vamos começar?

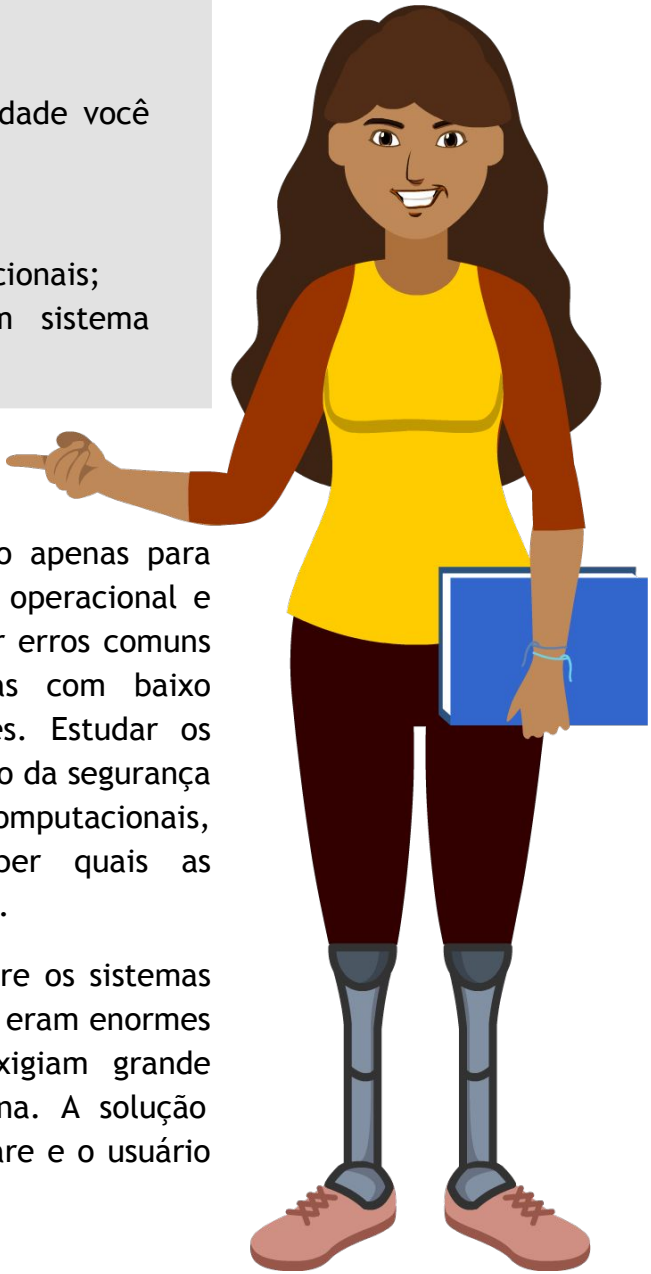
Unidade 1

SISTEMAS OPERACIONAIS - HISTÓRIA E PLATAFORMAS

Objetivos de aprendizagem

Esperamos que, ao final do estudo desta unidade você seja capaz de:

- Operar sistemas operacionais;
- Compreender a arquitetura de Sistemas Operacionais;
- Distinguir os módulos de gerência de um sistema operacional.



1.1 Introdução aos Sistemas Operacionais

Estudar os sistemas operacionais é importante não apenas para compreender os mecanismos que compõem um sistema operacional e permitir a execução de tarefas, mas também para evitar erros comuns de programação, que podem resultar em programas com baixo desempenho ou, até mesmo, a perda de informações. Estudar os sistemas operacionais é importante, também, pelo aspecto da segurança e principalmente para quem irá administrar os sistemas computacionais, pois para defender um sistema é necessário saber quais as vulnerabilidades e as possíveis falhas que o sistema possui.

Começaremos o nosso caminho de aprendizado sobre os sistemas operacionais relembrando que os primeiros computadores eram enormes e possuíam uma programação bastante complexa. Exigiam grande conhecimento de *hardware* e de linguagem de máquina. A solução encontrada foi a criação de uma camada entre o hardware e o usuário do sistema computacional.

Essa camada foi denominada Sistema Operacional e serviu para o encapsulamento das interfaces de *hardware*. Dessa forma, a interação com o computador se tornou mais fácil, confiável e eficiente.

De acordo com Machado (2011), "o Sistema Operacional tem por objetivo funcionar como uma interface entre o usuário e o computador, tornando sua utilização mais simples, rápida e segura".

Neste mesmo alinhamento, Maziero (2017, p. 1) define um sistema computacional e um sistema operacional da seguinte forma:

> "um sistema de computação é constituído basicamente por *hardware* e *software*. O *hardware* é composto por circuitos eletrônicos (processador, memória, portas de entrada/saída, etc.) e periféricos eletro-óptico-mecânicos (teclados, mouses, discos rígidos, unidades de disquete, CD ou DVD, dispositivos USB, etc.). Por sua vez, o software de aplicação é representado por programas destinados ao usuário do sistema, que constituem a razão final de seu uso, como editores de texto, navegadores Internet ou jogos. Entre os aplicativos e o *hardware* reside uma camada de *software* multifacetada e complexa, denominada genericamente de Sistema Operacional."

Para Tanenbaum (2010), o sistema operacional realiza basicamente duas funções não relacionadas:

> [...] "fornecer aos programadores de aplicativos (e aos programas aplicativos naturalmente) um conjunto de recursos abstratos claros em vez de recursos confusos de *Hardware* e gerenciar esses recursos de *Hardware*."

Assim, podemos compreender que um sistema computacional deve permitir a solução de problemas e a execução adequada de programas dos usuários.

Avançando na definição apresentada por Maziero, podemos considerar que um sistema computacional é composto por três componentes básicos e fundamentais: as aplicações, os usuários e o hardware.

O primeiro componente, as aplicações, definem como os recursos do sistema serão utilizados para resolver os problemas computacionais dos usuários. São exemplos de aplicações: compiladores, banco de dados, programas comerciais e jogos.

O segundo componente, os usuários, são os utilizadores do sistema computacional. São exemplos de usuários: as pessoas, as máquinas e outros computadores ou sistemas computacionais.

O terceiro componente, o hardware, são os recursos básicos de computação e se dividem em três subsistemas básicos: Unidade Central de Processamento, Memória Principal e Dispositivos de Entrada e Saída. A Figura 1 demonstra essa interação entre os usuários, aplicações e o hardware, sempre mediada pelo Sistema Operacional.

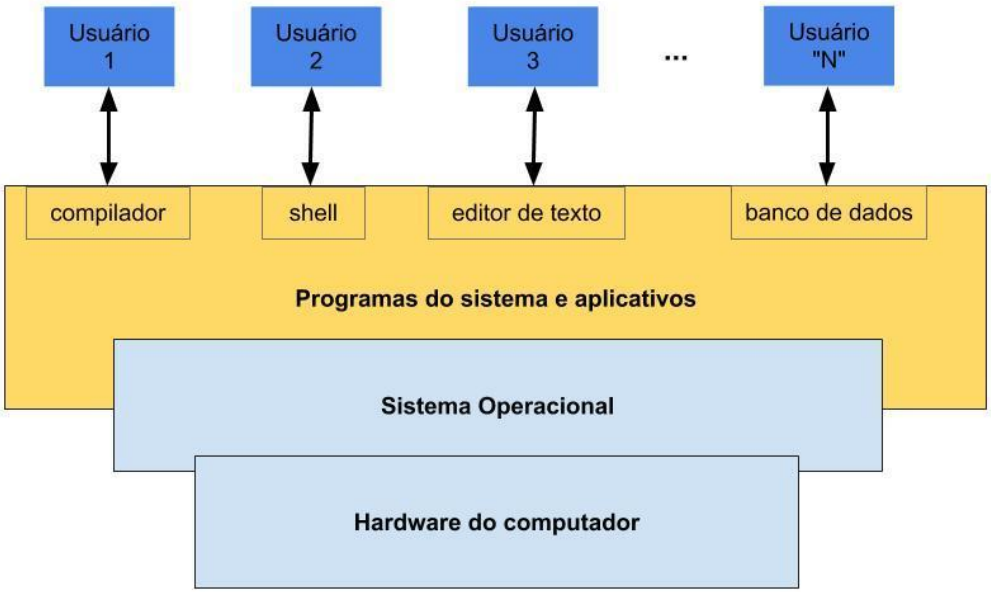


Figura 1 - Componentes básicos de um sistema computacional.
Fonte: traduzida e adaptada de http://www.komputersalatiga.com/2019/08/complete-computer-component-image-and-its-functions.html

O sistema operacional atua como uma camada de *software* entre os programas aplicativos dos usuários finais e o *hardware*. Ele é uma estrutura complexa de *software*, bastante ampla, que incorpora aspectos de alto nível, como a interface gráfica e os programas utilitários, e de baixo nível, como a gerência de memória e os *drivers* de dispositivos.

Um sistema operacional pode ser multitarefas e multiusuários, como demonstrado na Figura 1. Neste caso, ele deverá permitir que cada processo e usuário utilizem os recursos de *hardware* e *software* de forma independente, sem se preocupar com os outros *softwares* e usuários que estão utilizando o sistema ao mesmo tempo.

A função de um sistema operacional pode ser estabelecida por meio de duas abordagens: *top-down* (de cima para baixo) ou *bottom-up* (de baixo para cima). A abordagem *top-down* considera que a partir da máquina real (*hardware*) podemos criar uma máquina estendida ("abstrata" ou "virtual") onde o sistema operacional é uma extensão do *hardware* que implementa uma interface para as aplicações (programas). A visão *bottom-up* descreve que o sistema operacional é um controlador de recursos do sistema, ou seja, gerencia os recursos de *hardware* disponíveis para as aplicações.

A finalidade principal de um sistema operacional pode ser sintetizada em duas palavras-chave: “abstração” e “gerência”.

A **abstração de recursos** é necessária para facilitar o acesso aos recursos de *hardware* de um sistema de computação, já que esta pode ser uma tarefa complexa considerando as características específicas que cada dispositivo físico pode apresentar, além da complexidade de suas interfaces. Essa abstração pode tornar os aplicativos independentes do *hardware*.

A **gerência de recursos** é importante para definir a prioridade de acesso dos aplicativos sobre os recursos de *hardware*, quando dois aplicativos necessitam acessar o mesmo recurso, por exemplo. Assim é o Sistema Operacional que deve definir a prioridade de acesso (gerenciar políticas) resolvendo eventuais disputas e conflitos. Sobre esses aspectos de abstração e gerência de recursos do sistema operacional, Maziero (2017, p.4) reforça que:

> um sistema operacional visa abstrair o acesso e gerenciar os recursos de *hardware*, provendo aos aplicativos um ambiente de execução abstrato, no qual o acesso aos recursos se faz através de interfaces simples, independentes das características e detalhes de baixo nível, e no qual os conflitos no uso do *hardware* são minimizados.

Desta forma, podemos compreender o sistema operacional como uma camada transparente para o usuário permitindo que o computador seja utilizado de forma conveniente e eficiente ocultando a complexidade do *hardware*.

A Figura 2 demonstra melhor essa interação entre o usuário (e seus *softwares* e aplicações) e o *hardware* do computador (dispositivos físicos) mediado pelo sistema operacional.

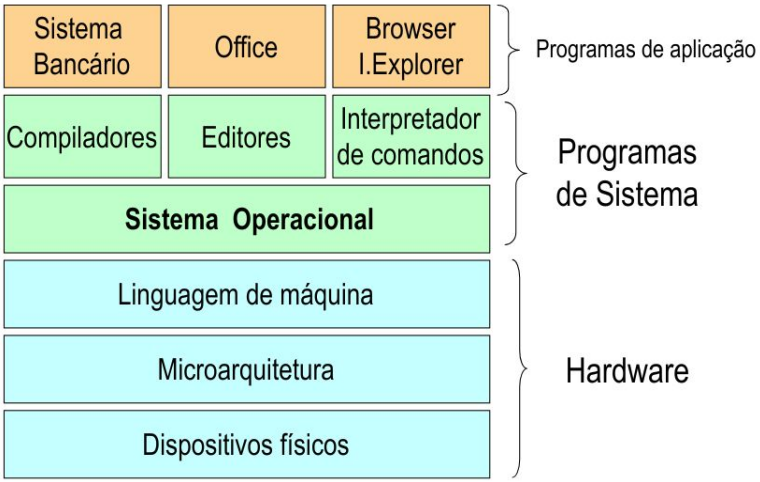


Figura 2 - Sistema operacional - Interface entre o usuário e o hardware.
Fonte: elaborado pelo autor.

A Figura 3 permite compreender que o sistema operacional também é responsável por oferecer interfaces padronizadas de acesso ao hardware e permitir uma visão homogênea dos dispositivos de hardware. O usuário clica em um arquivo, na sua aplicação, e o sistema operacional faz a interface de acesso ao hardware e aos dados no disco de forma transparente.

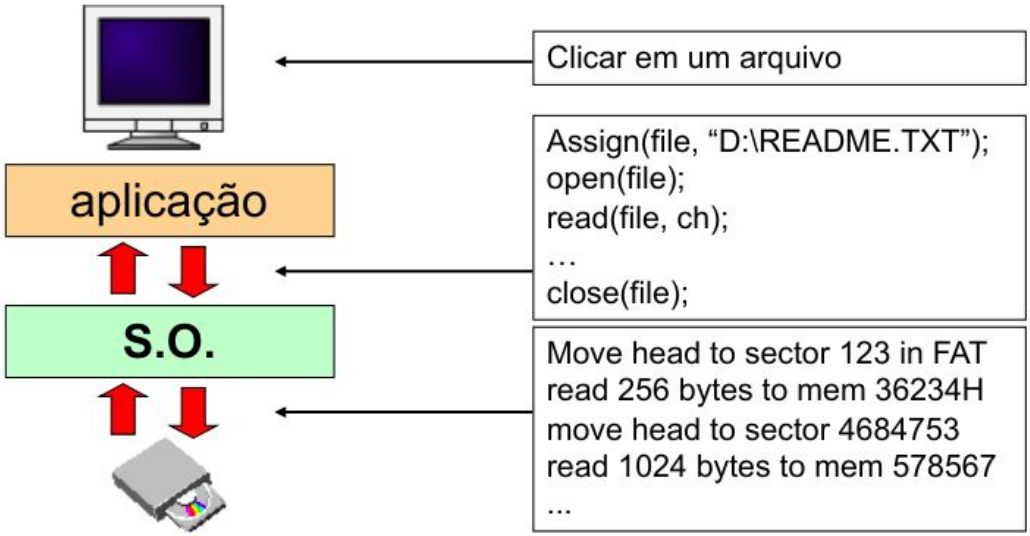


Figura 3 - Sistema operacional como máquina estendida (abstrata).
Fonte: elaborado pelo autor.

O sistema operacional como gerente de recursos faz a gestão do uso do processador, do espaço em memória, do acesso a arquivos, à conexões de rede e a dispositivos externos, sempre buscando evitar conflitos.

Cada sistema operacional oferece serviços e programas específicos, porém apresentam funções em comum. Vejamos algumas:

- **Execução de programas:** capacidade de carregar, executar e finalizar um programa;
- **Operações de I/O (entrada e saída):** deve fornecer meios para controlar arquivos ou dispositivos de I/O;
- **Manipulação do sistema de arquivos:** ler, gravar, criar e excluir arquivos;
- **Comunicação:** comunicação entre processos;
- **Detecção de erros:** indicar falhas de CPU, memória ou dispositivos de I/O e tomar medidas adequadas;
- **Alocação de recursos:** gerenciar recursos de memória, CPU ou dispositivos I/O;
- **Contabilização:** manter um registro dos usuários que utilizam os recursos do computador referente a quantidade e que tipo de recursos;
- **Proteção:** A proteção visa garantir que todo acesso aos recursos do sistema seja controlado, evitando conflitos e integridade dos dados.

Podemos desta forma, considerar que um sistema operacional deve possuir características desejáveis.

Vejamos a tabela a seguir para compreendermos melhor quais características são essas:

Característica	Definição	Exemplo
Concorrência	Existência de várias atividades ocorrendo paralelamente.	Execução simultânea de "*jobs*" (tarefas), E/S paralela ao processamento.
Compartilhamento	Uso coordenado e compartilhado de recursos de *Hardware* e *Software*.	Custo de equipamentos, reutilização de programas, redução de redundâncias, etc.
Armazenamento de dados	Capacidade de armazenamento a longo prazo.	Preservação das informações.
Não determinismo	Atendimento de eventos que podem ocorrer de forma imprevisível.	Atender a demandas inesperadas do *hardware*, *softwares* ou dos usuários.
Eficiência	Baixo tempo de resposta, pouca ociosidade da CPU e alta taxa de processamento.	Ampliar o desempenho do SO.
Confiabilidade	Pouca incidência de falhas e exatidão dos dados computados.	Ampliar o desempenho do SO.
Mantenabilidade	Facilidade de correção ou incorporação de novas características.	Facilidade de evolução.
Pequena dimensão	Simplicidade e baixa ocupação da memória.	Facilidade de uso.

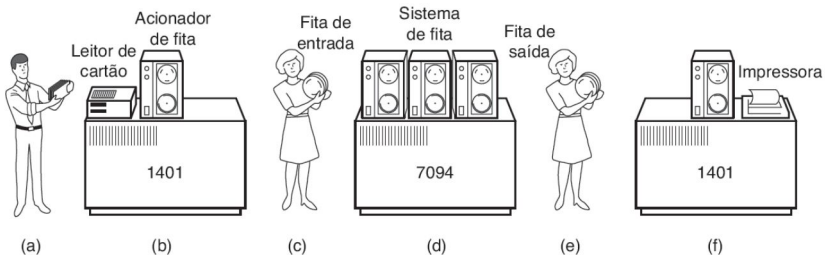
1.2 História dos Sistemas Operacionais

Não podemos aprofundar o tema de sistemas operacionais sem antes fazer uma rápida revisão sobre o seu desenvolvimento histórico. Esse processo permitirá entender como surgiram determinadas características e padrões de design que continuaram se desenvolvendo nas décadas seguintes. Conhecer os fatores que motivaram diferentes desenvolvimentos podem ajudar a prever e prevenir problemas.

Quando os Sistemas Operacionais surgiram, o usuário era o programador e o operador da máquina ao mesmo tempo. Existia muita interação humana no processamento das tarefas. Assim como nas arquiteturas de hardware, os Sistemas Operacionais também passaram por um processo evolutivo, classificado em gerações ou fases.

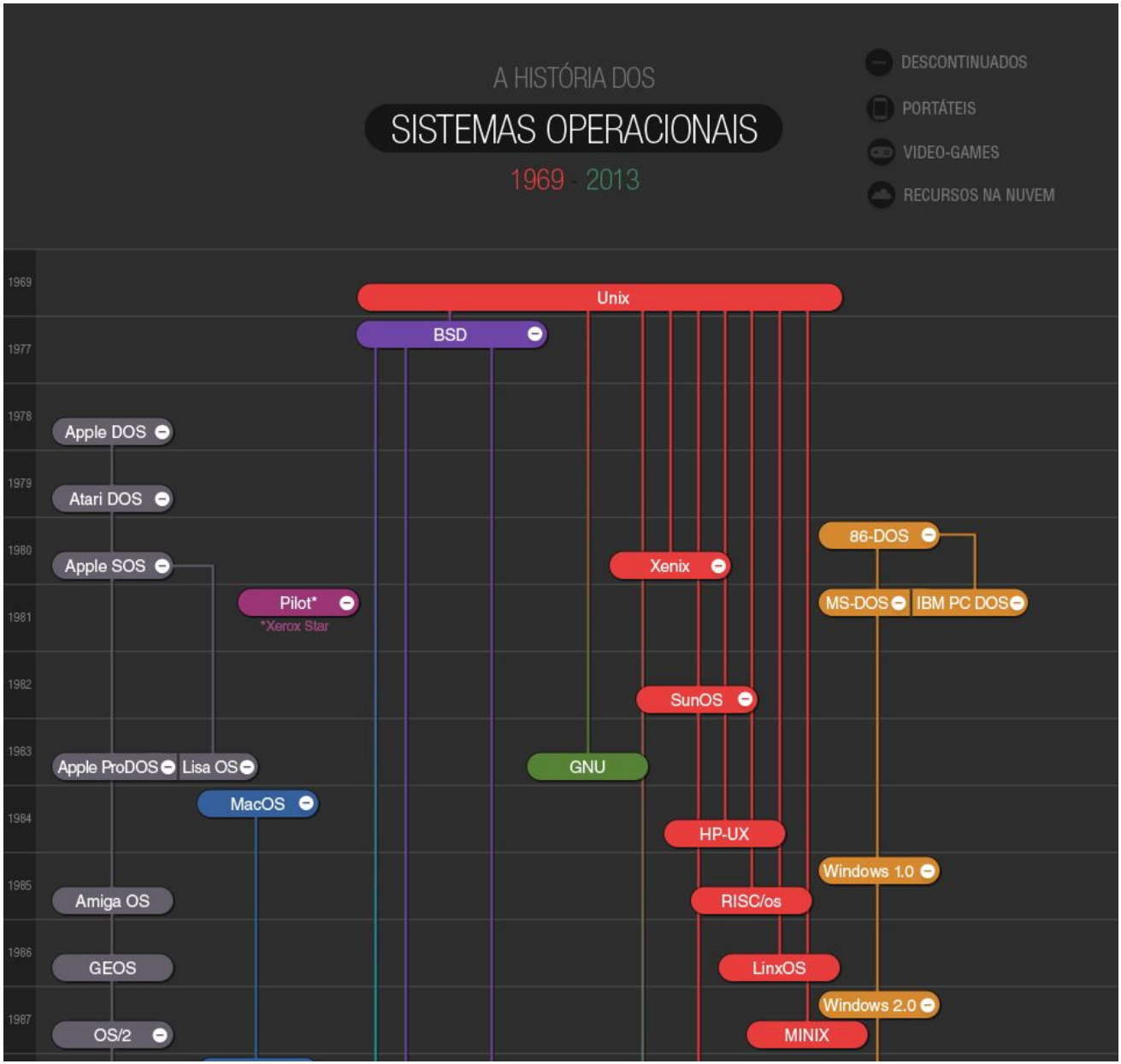
Vejamos a evolução histórica dos Sistemas Operacionais, agrupada em 5 fases:

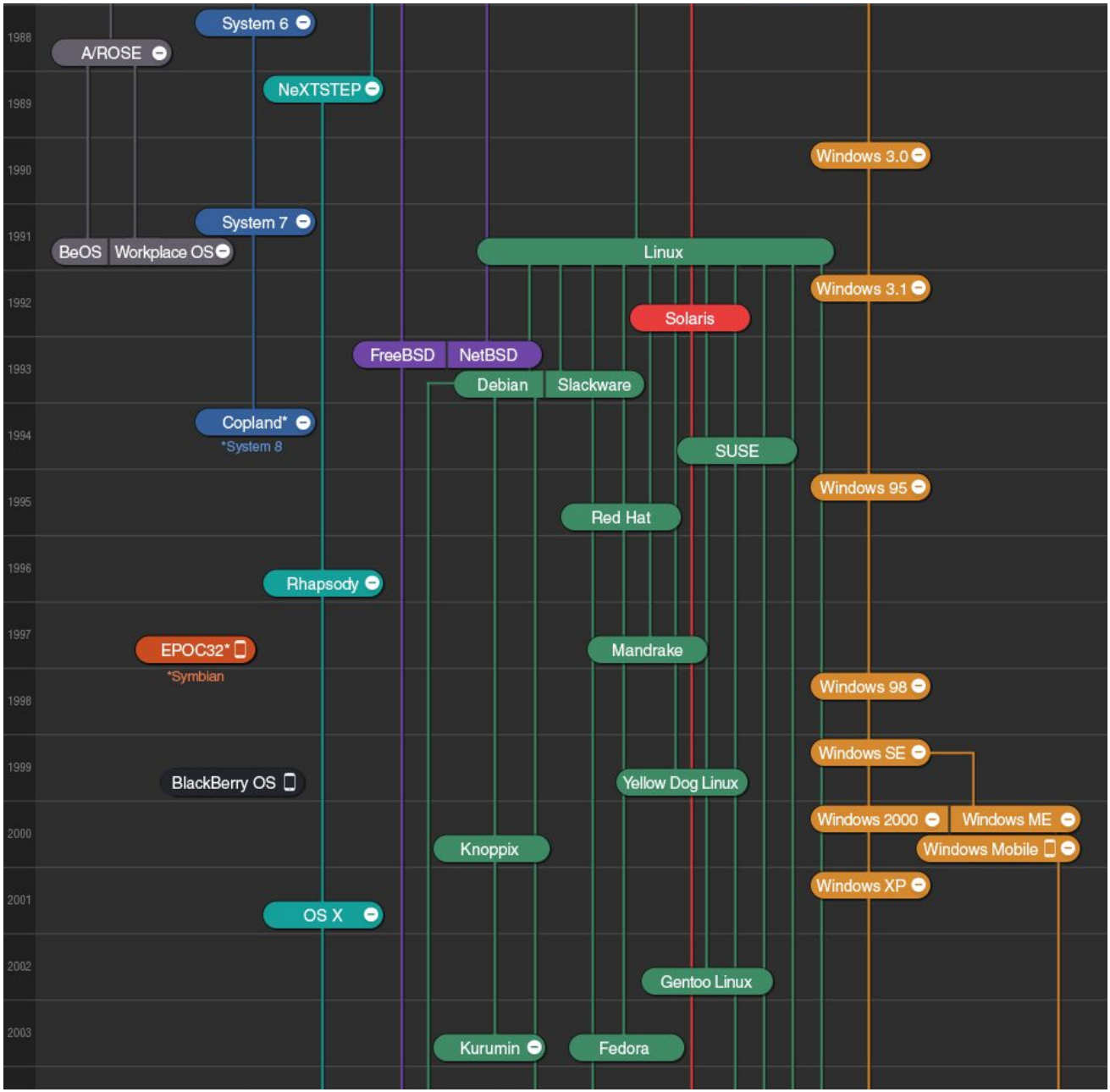
Fase Inicial (Fase 0) *Computadores são uma ciência experimental e exótica. Não precisa de sistema operacional*	• Programação através de "*plugs*". • Usuário presente todo o tempo e toda atividade é sequencial. • Conjuntos de cartões manualmente carregados para executar os programas. • Primeiras bibliotecas, utilizadas por todos. • O usuário é programador e operador da máquina ao mesmo tempo. **Problema: muita espera!** • Usuário tem que esperar pela máquina ... • Máquina tem que esperar pelo usuário ... • Todos têm que esperar pela leitora de cartões!

1ª Fase *Altos Preços. Computadores são caros. Pessoas são baratas.*	• SO surge com o objetivo básico de automatizar a preparação, a carga e a execução de programas. • SO torna a utilização do computador mais eficiente, desacoplando as atividades das pessoas das atividades do computador. • Mais tarde: otimização do uso dos recursos de *hardware* pelos programas. • SO funciona como um monitor *batch*, continuamente carregando um *job*, executando e continuando com o próximo *job*. Se o programa falhasse, o SO salvava uma cópia do conteúdo de memória para o programador depurar. **Exemplo de um sistema em lotes (*batch*) antigo:**  (a) Os programadores levam os cartões para o 1401. (b) O 1401 grava os lotes de tarefas nas fitas. (c) O operador leva a fita de entrada para o 7094. (d) 7094 executa o processamento. (e) O operador leva a fita de saída para o 1401. (f) 1401 imprime as saídas Figura 4 - Sistema em Lote Fonte: traduzido de: https://es.slideshare.net/jairakolatronic/evolucion-de-los-sistemas-operativos-32711361

2ª Fase *Produtividade - Custo/Benefício* *Computadores são rápidos; pessoas são lentas; ambos são caros.*	• "*Timesharing*" interativo: permitir que vários usuários utilizem a mesma máquina simultaneamente. • Um terminal para cada usuário. • Manter os dados "*on-line*": utilização de sistemas de arquivos estruturados. **Problema:** • Como prover tempo de resposta razoável?
3ª Fase *Produtividade - Custo/Benefício* *Computadores são baratos; pessoas são caras. Dar um computador para cada pessoa.*	• *Workstation* pessoal (SUN - Stanford University Network, Xerox Alto) • *Apple II* • *IBM PC* • *MacIntosh*
4ª Fase *Popularização Computadores Pessoais (PCs) em todo o planeta.*	• Redes possibilitam aparecimento de novas aplicações importantes. **Problemas:** • As pessoas ainda continuam esperando por computadores • Vírus, *worms*, *hackers*...

Os sistemas operacionais estão presentes nos mais diversos dispositivos, desde computadores pessoais até dispositivos móveis. A história do desenvolvimento dos sistemas operacionais pode ser estudada por meio de uma linha do tempo.


A HISTÓRIA DOS
SISTEMAS OPERACIONAIS
1969 - 2013
DESCONTINUADOS
PORTÁTEIS
VIDEO-GAMES
RECURSOS NA NUVEM
1969
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
Unix
BSD
Apple DOS
Atari DOS
Apple SOS
Pilot*
*Xerox Star
Xenix
86-DOS
MS-DOS
IBM PC DOS
SunOS
GNU
Apple ProDOS
Lisa OS
MacOS
HP-UX
Windows 1.0
Amiga OS
RISC/os
GEOS
LinxOS
Windows 2.0
OS/2
MINIX


1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
System 6
A/ROSE
NeXTSTEP
Windows 3.0
System 7
BeOS
Workplace OS
Linux
Windows 3.1
Solaris
FreeBSD
NetBSD
Debian
Slackware
Copland*
*System 8
SUSE
Windows 95
Red Hat
Rhapsody
EPOC32*
*Symbian
Mandrake
Windows 98
Windows SE
BlackBerry OS
Yellow Dog Linux
Windows 2000
Windows ME
Windows Mobile
Knoppix
Windows XP
OS X
Gentoo Linux
Kurumin
Fedora

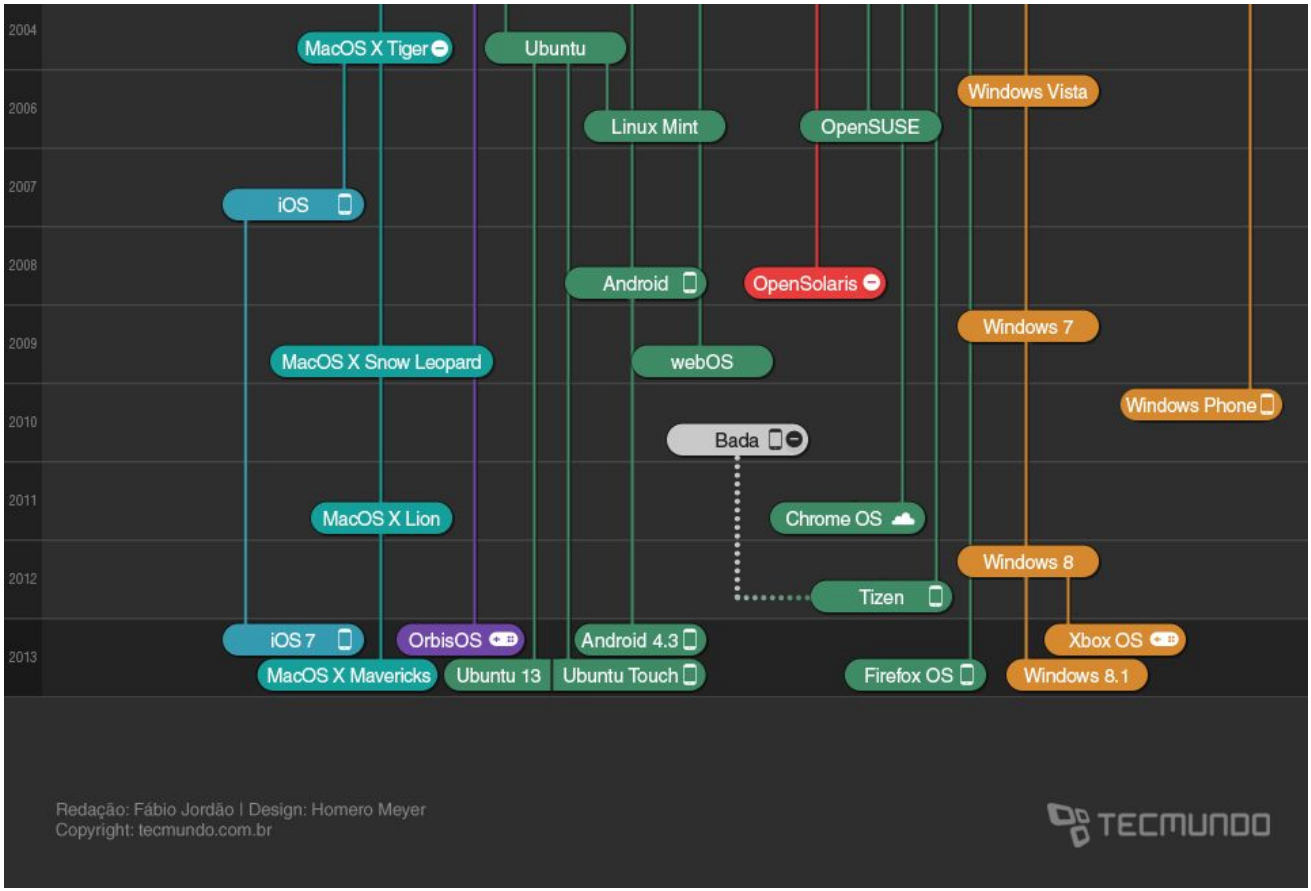


Figura 5 - Linha do tempo dos Sistemas operacionais.
Fonte: Site Techmundo - Disponível em:
https://www.tecmundo.com.br/sistema-operacional/2031-a-historia-dos-sistemas-operacionais-ilustracao-.htm

Como surgiram os sistemas operacionais?

Porque eles são tão importantes?

Para entender um pouco mais sobre sistemas operacionais e para responder às questões acima, assista ao vídeo que selecionamos. Ele trata da história dos sistemas operacionais.

Acesse:
https://www.youtube.com/watch?v=9rC9GilX1Io

1.3 História da Mobilidade

Nos últimos anos, grande parte do desenvolvimento no mundo da computação voltou-se para o modelo de computador representado, genericamente, por dispositivos móveis. O interesse por essas plataformas cresceu tanto que é necessário abordar a questão buscando sempre perceber as semelhanças com o restante dos equipamentos de informática.

Para fazer isso, no entanto, primeiro é necessário abordar a definição: o que é um dispositivo móvel ? Quais são os limites da sua definição?

Que fronteiras podem ser definidas? É difícil encontrar limites claros e rígidos para o que este conceito engloba.

Precisamos considerar que um dispositivo móvel é um computador concebido não apenas em nível de *hardware*, mas também, com design e interface de usuário, projetado para atender e organizar as demandas do usuário em suas tarefas diárias e cotidianas. Certamente, essa definição deixará alguma área cinzenta, pois está longe de ser perfeita e permitirá alguma ambiguidade. Um exemplo desse processo está nas versões mais recentes de alguns ambientes para dispositivos móveis (windows 8, gnome e *unity* do GNU/Linux) que buscam unificar a experiência do usuário incorporando conceitos dos *Desktop*.

Para começar nossa caminhada pela história da mobilidade e entender o surgimento dos sistemas operacionais desses equipamentos móveis, vamos refletir e tentar responder a essa pergunta inicial: você imagina quantos celulares (dispositivos móveis) há no Brasil?

De acordo com dados da Anatel, o Brasil terminou outubro de 2018 com 233,3 milhões de celulares e a densidade de 111,34 celulares para cada 100 habitantes. Incrível, não?

Esse resultado é fácil de compreender considerando que a grande maioria desses aparelhos faz muito mais do que chamadas telefônica. Foi-se o tempo em que um celular servia apenas para substituir os telefones, com a facilidade de podermos carregá-los conosco. Os telefones mais inteligentes, ou smartphones, são verdadeiros computadores de bolso, com capacidade de processamento superior a muito dos grandes computadores da década de 1980. Uma pesquisa da Fundação Getúlio Vargas (FGV), publicada em abril de 2016, apontou que existem 168 milhões de *smartphones* no Brasil. Isso quer dizer que, daqueles mais de 250 milhões de celulares que existem no País, 66% são máquinas inteligentes.

Certamente, essa proporção irá aumentar, devido à natural e gradual queda de preços dos equipamentos mais sofisticados.

a) Uso de Celulares e Smartphones no Brasil

Hoje, graças aos avançados dos sistemas que fazem esses aparelhos funcionarem, os chamados sistemas operacionais, dispomos de uma enormidade de aplicativos que nos permitem realizar diversas ações. Atualmente, podemos utilizar esses equipamentos para pagar contas, pedir comida, fazer compras, jogar, acessar redes sociais e trocar mensagens, assistir TV, filmes e ouvir músicas. Com eles podemos ainda ler livros e revistas, tirar e enviar fotos e filmes, e até mesmo chamar um táxi. Mas as facilidades não terminam aqui.

Além dessas funcionalidades descritas, podemos ainda controlar outros equipamentos, consultar a previsão do tempo, checar um trajeto de deslocamento e receber informações sobre o trânsito. Tudo está se integrando em um único equipamento. Esse processo é chamado de convergência digital.

Na verdade, essa lista cresce diariamente, pois sempre há alguém pensando em uma nova utilidade para esses aparelhos. Passamos cada vez mais a integrar a tecnologia ao nosso cotidiano e a depender de equipamentos móveis para muitas de nossas atividades diárias: lembrar o que temos de fazer, avisar com quem temos de falar e escolher o melhor caminho a seguir.

Não podemos esquecer que cada equipamento desses possui um hardware com inúmeras funcionalidades e necessita de um sistema operacional para coordenar e controlar os aplicativos e *softwares*, servindo de interface entre os usuários e os dispositivos disponíveis.

O acesso fácil e contínuo a redes sociais e aplicativos de troca de mensagens mudou os hábitos de milhões de pessoas, que passaram a ficar conectadas com amigos – da vida real ou virtuais – mesmo estando a muitos quilômetros de distância.

A dependência é tão grande que muita gente diz se sentir “sem roupas” quando esquece o celular em casa, e isso não deixa de ser verdade!

Vício Digital

Quando a dependência dos dispositivos móveis e a necessidade de estar conectado 24 horas por dia ultrapassa os limites do razoável, trata-se de vício digital. Mas o que seria razoável? Difícil de dizer. O fato é que, se a utilização dos dispositivos móveis, ao invés de ajudar, começar a atrapalhar a vida e o trabalho de uma pessoa, esse deve ser um sinal de alerta para que seus hábitos sejam revistos.

Vamos ler um pouco mais sobre o tema?

Visite:
https://repositorio.ufpb.br/jspui/handle/123456789/1867

b) Outros Equipamentos Móveis

Falamos muito dos celulares e *smartphones*, mas há uma série de outros equipamentos móveis. Observe a tabela a seguir com um pequeno exemplo descrevendo os equipamentos móveis mais comuns disponíveis atualmente.

Tablets	Terminais de pagamento	*Smartwatches*
		
Imagem disponível em: http://shopfacil.vteximg.com.br/arquivos/ids/11848376/image-d86ad6240abb4a239f7ccc68db0250dd.jpg?v=636595737675500000	Imagem disponível em: http://melhormaquinadecartao.com/wp-content/uploads/10-Melhores-Ma%CC%81quinas-de-Carto%CC%83es-Pelo-Celular-Aplicativo-Android-e-iOS-.png	Imagem disponível em: https://www.altonivel.com.mx/assets/images/Estructura_2016/Tecnologia/smart-watch.jpg
Maiores e mais confortáveis para se utilizar com aplicações que exigem muita digitação ou com as que têm telas mais complexas, os *tablets* utilizam, basicamente, os mesmos aplicativos que os *smartphones*, pois são baseados nos mesmos sistemas operacionais.	Máquinas móveis conectadas em rede, os terminais de pagamento de lojas, restaurantes e postos de combustível têm sistemas internos, programas e aplicativos que os fazem executar funções específicas, por isso, também podem ser considerados dispositivos móveis, embora sejam de utilização mais restrita.	Evoluindo em direção a miniaturização e integração dos equipamentos às pessoas – dispositivos para "vestir" –, os *smartwatches* (relógios inteligentes) já estão disponíveis no mercado. Embora ainda limitados a servir de conexão aos *smartphones*, esses relógios deverão tornar-se equipamentos independentes em breve, assumindo funções que hoje são realizadas pelos *smartphones* e *tablets*.

O crescimento no número de dispositivos que empregam sistemas operacionais não para. E o futuro promete ser ainda mais cheio de dispositivos conectados! Vamos entender o por quê?

c) Internet das Coisas

Criado em 1999, pelo *Massachusetts Institute of Technology* (MIT), o conceito da "Internet das Coisas" está cada vez mais se desenvolvendo. O MIT prevê que todos (ou quase todos) os objetos existentes, como carros, lâmpadas e canetas estarão conectados na próxima década. Estudos realizados pelo instituto, dão conta que cada um de nós está em contato com cerca de 1.000 a 5.000 objetos nesse atribulado dia a dia moderno.

Pense em quanta informação pode ser gerada se soubermos em tempo real (conectados) quais são e onde estão todos esses objetos e equipamentos?
Imagine acessar esses dispositivos em tempo real. Verificar se é hora de manutenção, ou controlá-los para coletar dados e executar determinadas tarefas.
Ou saber o que está ocorrendo em torno deles em tempo real, ou até mesmo se um produto dentro da geladeira está vencendo ou com validade expirada. Incrível, não? Isso tudo já é possível!

Imagem disponível em: http://guiadecompras.casasbahia.com.br/imagens/2017/04/internet-coisas-dia-a-dia.jpg

Cada vez mais baratos e populares, a conectividade sem fio com esses dispositivos já é uma realidade. Seus pequenos *chips* e processadores, integrados a essa conectividade real, ampliam as possibilidades de uso e emprego. O limite está no *hardware*, no sistema operacional empregado, nos *softwares* e aplicativos, e é claro na nossa imaginação.

De onde vieram os smartphones tão populares e indispensáveis para nós?

Vamos assistir a dois vídeos que irão nos ajudar a compreender o surgimento dos dispositivos móveis. Como surgiu o celular? Como evoluiu a cultura da mobilidade e da conectividade em dispositivos móveis ?

Para saber um pouco mais e para entender como essa evolução aconteceu, separamos dois vídeos interessantes.

O primeiro é do CANAL NOSTALGIA no YOUTUBE narrando à evolução do celular.
Acesse: https://www.youtube.com/watch?v=lNbjAiEUQQU

O segundo é do Discovery Ciência e conta a história do celular.
Acesse: https://www.youtube.com/watch?v=yjiB_Yw05RE

Agora que já conhecemos a história dos sistemas operacionais e os principais trechos da história da mobilidade, vamos observar, de forma mais detalhada, a evolução tecnológica das telecomunicações que nos trouxe até o estágio atual. Sim, entender como a conectividade e a possibilidade de acesso à Internet nos dispositivos móveis são importantes para compreender como ocorreu a evolução dos dispositivos móveis e seu sistema operacional. Para isso, vamos seguir os caminhos das duas tecnologias que deram origem aos dispositivos móveis modernos: a telefonia celular e a computação móvel.

d) Telefonia Celular

A ideia de ter um telefone móvel nasceu em 1947. No entanto, à época, os recursos tecnológicos disponíveis não permitiram criar nada além de telefones que funcionavam como radiocomunicadores, e só eram viáveis em veículos automotores e embarcações, devido, principalmente, ao volume e ao peso de seus transmissores e baterias.

Durante a Segunda Guerra Mundial, versões rudimentares desses aparelhos portáteis de radiocomunicação foram carregadas pelas tropas americanas. Em cada equipe de campo havia um soldado responsável pelas comunicações que levava o equipamento em uma grande e pesada mochila. O alcance e a duração das baterias eram limitados e não permitiam um funcionamento regular por muito tempo. Mesmo assim, essa “mobilidade” inspirou os pesquisadores a pensarem em equipamentos móveis para uso no dia a dia.

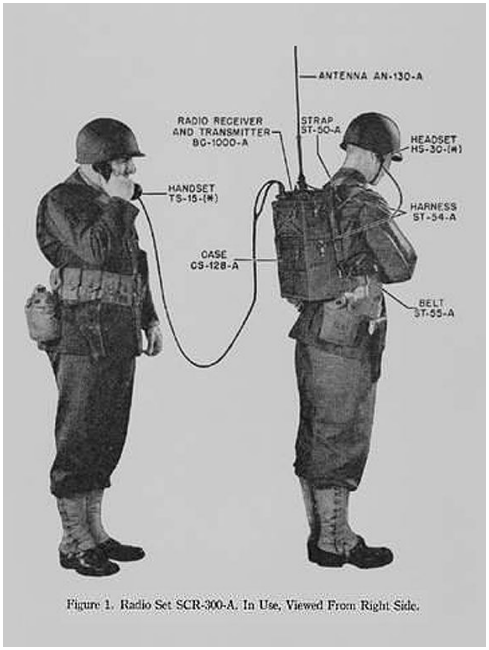


Figura 6 - Rádio comunicador portátil - SCR-300 - Inspiração para o celular.
Fonte: Wikipedia. Disponível em
https://es.wikipedia.org/wiki/SCR-300#/media/File:Scr300.png

Em 1973, os primeiros testes de uma rede de telefonia celular deram certo. Nesse momento, foi definido o modelo de funcionamento que é utilizado até hoje, com uma rede de estações – chamadas rádio base – que fornece uma cobertura de sinal para os aparelhos. Dessa forma, um aparelho celular deve estar ao alcance de uma dessas estações para que funcione. Somente dez anos após os testes iniciais, o primeiro aparelho celular foi comercializado. Naquela época, ele pesava cerca de 1 kg e tinha 30 cm de altura. O equipamento se chamava DynaTAC e era produzido pela Motorola.

Fonte: Techmundo. Disponível em <https://www.dinheirovivo.pt/wp-content/uploads/2017/04/dyna.jpg >

Fonte: Dinheirovivo. Disponível em <https://s.glbimg.com/po/tt/f/original/2011/07/08/2007computex_e21forum-martincooper.jpg >

Figura 7 - Motorola Dynatac 8000X, um verdadeiro celular portátil.

Era inevitável que a tecnologia evoluísse, tornando os aparelhos menores e mais fáceis de utilizar. No início da década de 1990, foram lançados novos aparelhos que já podiam ser carregados por uma pessoa, sem grande esforço, em uma bolsa ou presos a um cinto. Essa revolução tecnológica popularizou a telefonia celular e provocou uma verdadeira explosão na quantidade de consumidores no mundo. No Brasil, essa tecnologia começou a chegar em 1990, mas ainda restrita e muito cara.

A ampliação da área de cobertura das operadoras e a redução dos valores dos serviços para o consumidor final, no Brasil, ocorreram quase dez anos depois. Foi a partir de 1998 que as redes brasileiras de telefonia celular sofreram uma grande expansão. Além da evolução dos aparelhos, era necessário que houvesse uma ampliação na oferta de serviços, por meio de novas funcionalidades.

A primeira dessas funcionalidades foi o serviço de mensagens curtas, conhecidas como SMS (*short message service*). A inovação das mensagens curtas foi criada pela Nokia, na Finlândia, também no início dos anos 1990, e foi logo adotada por outros fabricantes, o que contribuiu para a evolução dos aparelhos celulares, que passaram também a trocar mensagens de texto.

Com a modernização das redes e o aumento da velocidade de transmissão disponível, os fabricantes enxergaram um novo mundo de possibilidades por meio da transmissão de dados. Inicialmente, os próprios fabricantes e as operadoras colocavam certos conteúdos disponíveis para *download*, ainda em redes fechadas, mas não demorou muito para que o acesso à internet fosse disponibilizado para celulares.

No início da década de 2000, a internet passou por uma verdadeira explosão de crescimento de oferta e utilização em todo o mundo. A conexão dos celulares a essa rede proporciona a disponibilização de uma gama nunca imaginada de novos serviços.

Surgiu então, uma verdadeira corrida pelo fornecimento de serviços móveis. Vendia-se a ideia de que era possível trabalhar de qualquer lugar e de que os computadores pessoais estavam com os dias contatos. Na verdade, estávamos ainda muito longe disso, mas a velocidade com que a tecnologia evoluiu foi impressionante.

Para entender a evolução dos sistemas operacionais e dispositivos é preciso entender como a conectividade e a oferta de novos serviços surgiram. Com o passar do tempo, a tecnologia empregada na transmissão de sinais dos celulares evoluiu rapidamente. Com isso, iniciou-se uma verdadeira guerra entre os fabricantes de equipamentos para se estabelecer um novo padrão tecnológico.

A maioria dos celulares comercializados no início da década de 2000 utilizava duas tecnologias amplamente difundidas na América do Norte: a TDMA (*Time Division Multiple Access*) e a CDMA (*Code Division Multiple Access*). Como a CDMA era mais evoluída, foi a mais utilizada no início da implantação das redes de telefonia celular no nosso país.

Em paralelo, empresas de telefonia europeias desenvolveram um sistema chamado GSM (*Global System for Mobile Communications*), que, além de suportar padrões mais elevados de velocidade e qualidade na transmissão, trazia uma novidade chamada módulo SIM (*Subscriber Identity Module*), o conhecido chip, ou cartão, que identifica o assinante ou “dono" da linha telefônica.

Não demorou para o GSM se tornar o novo padrão mundial, principalmente porque os fabricantes viram uma grande oportunidade: com a tecnologia GSM, o aparelho não ficava mais “preso” à linha telefônica. Dessa forma, era possível que o usuário demorasse menos tempo para trocar de equipamento e até tivesse mais de um aparelho, bastando colocar seu cartão SIM no aparelho que desejasse utilizar. A tecnologia GSM pode ser considerada um marco na história da telefonia e foi responsável por tornar os aparelhos celulares objetos mais ligados à moda, ou seja, facilmente substituíveis pelo usuário. Hoje essa é a tecnologia mais utilizada no mundo.

As novas funcionalidades dos aparelhos e a crescente demanda por transmissão de voz e dados moveram o desenvolvimento de redes de transmissão de segunda geração, as chamadas 2G, mais adequadas para suportar o novo padrão de utilização.

A transformação da telefonia celular de, apenas, mais um canal de voz para um sistema mais completo de comunicação foi considerada o ponto de partida para a consolidação da computação móvel e o surgimento dos sistemas operacionais para dispositivos móveis.

A tecnologia dos aparelhos e as novas redes 3G e 4G tornaram-se ferramentas de uso diário de milhares de pessoas ao redor do mundo. Vimos até aqui que a evolução da telefonia celular proporcionou o acesso a voz e dados em um aparelho móvel. É nesse ponto de nossa História que as tecnologias começam a se confundir com a evolução da computação móvel. Vamos ver?

e) Computação Móvel

A computação móvel nasceu em meados de 1992, com o lançamento do handheld Newton pela Apple. Handheld era o nome dado aos aparelhos que podiam ser utilizados em uma mão, também chamados de PDA (*Personal Digital Assistant*) ou assistente pessoal digital.

O Newton tinha 1 MB de memória e tela sensível ao toque, uma inovação para a época. Apesar disso, não chegou a ser um sucesso. Embora muita gente falasse dele, suas vendas foram inferiores a mil unidades no primeiro ano de comercialização.

Além de ser considerado grande, pesado e caro, o aparelho não dispunha de aplicativos que justificassem sua compra pela maioria das pessoas. Este é o grande segredo dos dispositivos móveis: para ter sucesso, eles têm de disponibilizar aplicativos úteis para as pessoas. Aplicativos? Pense e responda: para que a maioria das pessoas utiliza seus telefones celulares hoje? Com certeza, não é para fazer ligações.

O mercado dos dispositivos móveis começou a mudar em 1996, com o lançamento do Palm Pilot, um PDA produzido pela empresa norte-americana U.S. Robotics, conhecida fabricante de componentes para computadores que, anos depois, criou outra empresa chamada Palm, dado o sucesso do produto.

O Palm Pilot teve várias versões e gerações, mas o que o tornou popular e responsável pela mudança nos hábitos dos consumidores foi a sua concepção: era um aparelho compacto, com uma inovadora interface por caneta, que permite fazer anotações à mão, de uma forma bastante eficiente. Devido ao êxito dessa nova plataforma, outros fabricantes lançaram dispositivos semelhantes, entre eles a HP, a Casio e a NEC. A Microsoft apostou na criação de um sistema operacional para ser o "padrão" dessas máquinas: o chamado Windows CE, que evoluiu para um sistema chamado Pocket PC no ano 2000.

Outros fabricantes, ainda no final do século XX, lançaram aparelhos que tinham pequenos teclados, o que limitava suas funcionalidades e determinava um tamanho mínimo. No entanto, a maquininha trazia agenda de contatos, agenda de compromissos, funcionalidades de alertas e lembretes, alguns jogos e uma interface com computadores para a sincronização dos dados, o que a tornou um sucesso! A vida estava bem melhor para quem tinha que trabalhar com mobilidade, pelo menos na questão de equipamentos de uso pessoal.

No início do ano 2000, os executivos mais modernos começavam a levar consigo dois aparelhos: um celular para fazer suas ligações e um handheld para manter sua agenda de compromissos e agenda de contatos, além de utilizar alguns aplicativos simples, como alarmes e lembretes.

Foi então que os fabricantes começaram a pensar em juntar as duas coisas. Será que os dois aparelhos não poderiam se tornar um só?

Uma das primeiras criações em direção à integração do celular com o handheld foi o Palm Pilot 7, que tinha todas as funcionalidades do tradicional PDA, mas com um telefone celular incorporado, com direito a anteninha e tudo mais.

No entanto, devido à mudança de tecnologia de transmissão de dados, que levou o GSM a ser o novo padrão mundial, esse aparelho teve vida curta e não chegou a ser comercializado no Brasil.

A partir dessa nova demanda para dispositivos móveis, os fabricantes de celulares do mundo todo começaram, então, a criar suas próprias máquinas inteligentes, com sistemas operacionais próprios.

Foi uma nova corrida, dessa vez para tentar definir um padrão para o smartphone. É importante que você conheça os sistemas que estão em uso, pois é para eles que se desenvolvem os aplicativos e são direcionadas as novidades. No decorrer desses anos, os dispositivos móveis evoluíram e os equipamentos mais antigos foram substituídos.

Em 2009, a pioneira Palm trocou seu Palm OS por um novo sistema operacional, chamado WebOS, cujas versões subsequentes foram utilizadas pela HP e LG. A Samsung criou, em 2010, o sistema operacional Bada, nome que significa "oceano" em coreano. O Bada foi incorporado à linha de smartphones Wave, com o objetivo de oferecer uma opção de baixo custo frente ao BlackBerry e ao iPhone. Nesse caminho, também desapareceram os sistemas Bada, Symbian e WebOS.

Estatísticas

Para saber mais sobre estatísticas de celulares no Brasil, acesse: http://www.teleco.com.br/ncel.asp

1.4 Sistemas Operacionais Desktop

Um sistema operacional para *desktop* pode ser entendido como o sistema operacional desenvolvido para um computador de um usuário final (pode ser um *notebook* ou um computador pessoal). É claro que existem sistemas operacionais para servidores que normalmente irão prover algum tipo de serviço específico (como servidor *web*, serviço de rede ou outra funcionalidade). Porém, a evolução do hardware dos computadores levou a uma aproximação dos sistemas operacionais.

Vamos conhecer um pouco sobre os sistemas operacionais mais utilizados em computadores e dispositivos móveis.

1.4.1 GNU / Linux

O Sistema operacional GNU / Linux surgiu quando Linus Torvalds, um estudante Finlandês de ciência da computação, anunciou uma versão prévia de um kernel para substituir o Minix em um *newsgroup Usenet* (comp.os.minix). A parte mais importante de um sistema operacional é o kernel. No SO GNU/Linux, o componente do kernel é o Linux. O restante do sistema consiste em outros programas, muitos dos quais escritos por, ou, para o projeto GNU. Como apenas o kernel sozinho não forma um sistema operacional utilizável, o correto é utilizar o termo "GNU/Linux".

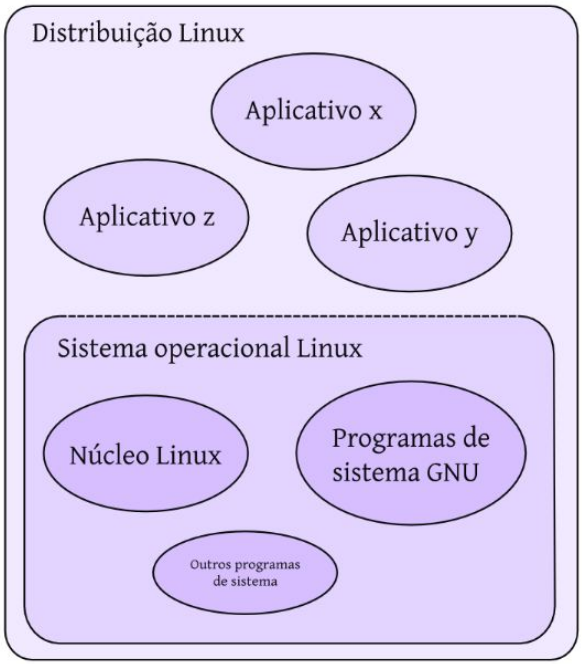


Figura 8 - Diagrama de uma distribuição GNU/Linux
Fonte: Wikipedia. Disponível em https://upload.wikimedia.org/wikipedia/commons/3/34/Diagrama_de_Venn_Linux_-_kernel%2C_so%2C_distribui%C3%A7%C3%A3o_%2B_explica%C3%A7%C3%B5es_%28pt-BR%29.png

O sistema operacional GNU/Linux é o resultado da contribuição de um grande número de empresas e pessoas, formando uma comunidade. Na verdade, o sistema GNU/Linux é um componente central, o qual é ramificado em vários produtos diferentes. Eles são chamados de distribuições.

As distribuições mudam completamente a aparência e o funcionamento SO de acordo com os objetivos da comunidade. As distribuições variam desde grandes e completos sistemas (mantidos por empresas) até pequenas distribuições que cabem em um cartão de memória USB ou rodam em computadores antigos (geralmente desenvolvidas por voluntários). As distribuições mais conhecidas são UBUNTU, DEBIAN, SUSE e REDHAT.

O GNU/Linux possui diversas interfaces gráficas e elas não podem ser confundidas com o Sistema Operacional, pois é apenas uma aplicação gráfica (interface) para tornar mais fácil o acesso ao sistema. As interfaces gráficas mais utilizadas são o KDE, GNOME e XFCE.

O que é Linux? - Conheça as principais distribuições!

Vamos conhecer um pouco mais sobre a história do GNU/LINUX e suas distribuições. O canal DIOLINUX apresenta esse sistema operacional, suas características e as principais distribuições.

Acesse:
https://www.youtube.com/watch?v=5nX4UFQt_JQ&t=110s

O site Distrowatch apresenta as principais distribuições GNU/Linux disponíveis.

Acesse:
https://distrowatch.com/

1.4.2 Windows

Os primeiros computadores não possuíam interface gráfica e o sistema era acessado apenas por comandos de texto. A usabilidade era bastante deficiente e complicada. A chegada da interface gráfica mudou essa perspectiva. A Microsoft iniciou com o sistema operacional MS-DOS (Microsoft Disk Operating System). As primeiras versões do Windows eram uma junção do MS-DOS com interface gráfica. A versão NT do Windows foi o primeiro sistema operacional da Microsoft a abandonar o MS-DOS. O Windows 95 (1995) apresentou o conceito plug and play (adição de periféricos), da barra ferramentas e do botão iniciar. O Windows 98 (1998), a partir do estouro da Internet, apresentou grandes novidades com o navegador IE4 (Internet Explorer) e o suporte ao padrão USB. O Windows 10 lançado em 2014 apresentou uma interface para dispositivos touchscreen e múltiplas áreas de trabalho.

Como surgiu o Windows?

É inegável a popularidade que o sistema operacional da Microsoft, o odiado e amado Windows, possui no segmento em que atua. O Windows não nasceu da forma como o conhecemos e nem com todos os recursos com os quais estamos familiarizados. Houve um processo gradual de evolução em que a Microsoft aprendeu quais eram as necessidades das pessoas e aperfeiçoou funcionalidades para equipar o seu programa.

Visite:
https://www.tecmundo.com.br/windows-10/64136-windows-1-windows-10-29-anos-evolucao-do-so-microsoft.htm

1.5 Sistemas Operacionais para Dispositivos Móveis e Mercado

Os dispositivos móveis (telefones celulares, smartphones, tablets e outros) ocupam cada vez mais tempo e espaço em nossas vidas. Com o advento da conectividade (internet), seus processadores estão mais velozes, há mais memória e um salto considerável no que diz respeito ao armazenamento foi dado.

Aquilo que era comum apenas aos computadores de mesa realizar, agora pode ser feito em qualquer lugar, a qualquer hora do dia, direto da palma da sua mão. Os dispositivos móveis, principalmente os smartphones, revolucionaram a nossa forma de acessar a internet e usar os aplicativos. Neste ambiente de revolução, estão também as empresas querendo "marcar sua posição" na guerra da mobilidade. Para os dispositivos móveis, seguindo a mesma linha do que ocorreu com os computadores pessoais, surgiram diversos sistemas operacionais. Vamos conhecer a seguir alguns deles.

1.5.1 Android

Projetado pela empresa Google, o Android baseia a maior parte de sua operação em software livre (um kernel Linux, uma máquina virtual Java e muitas das bibliotecas de sistema comuns ao GNU/Linux), adicionando uma camada de serviços proprietários. A estratégia do Google tem sido o oposto da Apple: em vez de fabricar seus próprios dispositivos, concede licenças para o uso do Android a praticamente todos os fabricantes de hardware, embarcando assim o seu sistema operacional na grande maioria dos modelos de smartphones e tablets.

A Google lançou, comercialmente, o sistema operacional *Android* em setembro de 2008, com o objetivo de oferecer ao mercado *smartphones* com grande variedade de recursos e com um custo mais baixo, já que a promessa era ter um sistema operacional sem custo, reduzindo o preço final dos aparelhos.

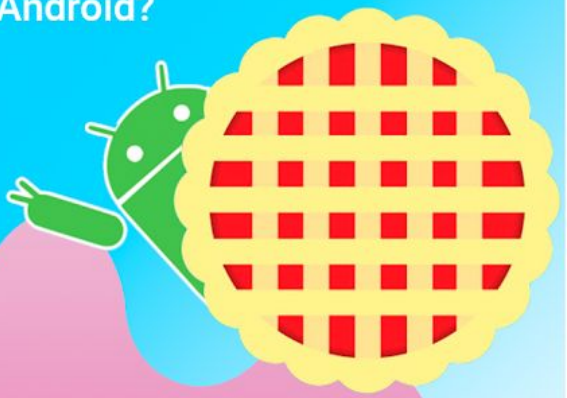
O *Android* foi logo adotado por grandes fabricantes, entre eles as coreanas Samsung e LG, a japonesa Sony e a americana Motorola, cuja divisão de equipamentos móveis pertenceu ao Google entre 2011 e 2014.

A utilização pelos grandes fabricantes, o custo baixo e a relativa facilidade de uso, colocaram, rapidamente, o *Android* à frente dos outros sistemas. Essas particularidades também incentivaram a realização de melhorias constantes, tanto na sofisticação dos dispositivos, quanto nas características e funcionalidades do próprio sistema operacional.

O sistema operacional *Android* passou por diversas versões desde o seu lançamento, todas acompanhadas de uma eficiente campanha de marketing, que associou os nomes das versões à doces.

Em 2008 e 2009, foram lançadas as primeiras duas versões, 1.0 e 2.0, foram chamadas de “A” e “B” respectivamente. Ainda em 2009, foram lançadas as versões *Cupcake* (bolo de caneca), *Donut* (rosquinha) e *Eclair* (bomba de chocolate), mantendo a ordem das letras, mas com a referência aos doces. Nos anos seguintes, tivemos a *Froyo* (sorvete de iogurte), *Gingerbread* (biscoito de gengibre), *Honeycomb* (favo de mel), *Ice Cream Sandwich* (um tipo de sorvete com biscoitos), *Jelly Bean* (bala de goma), *KitKat* (o famoso chocolate), *Lollipop* (pirulito), *Marshmallow*, e a *Nougat* (torrone), que é a versão 7.0 de agosto de 2016.

Você conhece as versões do sistema Android?



Este mês foi lançado ao público o **Android Pie**, a 9ª versão do sistema operacional. Aproveite a novidade para rever as diferenças e avanços de cada versão. Confere aí!

EVOLUÇÃO DO ANDROID

Android 1.0
Alpha

- Navegador Web simples;
- App Market para aplicativos;
- Suporte para câmera simples.

Android 1.6
Donut

- Interface para a programação de apps com reconhecimento de gestos;
- API de programação para uso de *text-to-speech*.

Android 2.1
Eclair

- Suporte para conexão Bluetooth 2.1;
- Adição e sincronização de várias contas;
- Flash, zoom digital e efeitos de cor nas fotos.

Android 2.3
Gingerbread

- Videochamadas;
- Suporte a NFC;
- Suporte a sensores de movimentos;
- Suporte a aparelhos com câmeras frontais.

Android 3.0
Honeycomb

Versão do Android projetada especialmente para tablets.

Android 4.0
Ice Cream Sandwich

- Controle de limite de dados utilizados;
- Android Beam – compartilhamento de dados por NFC;
- Tela inicial personalizada com organização de widgets;
- Unificou o SO de tablets e smartphones.

Figura 9 - Versões do SO *Android*
Fonte: Ilhasoft. Disponível em http://www.ilhasoft.com.br/wp-content/uploads/2018/08/Infografico-android.jpg

A versão mais atual do *Android* é a *Pie* (9.0). A versão *Oreo* (8.0) apresentou funcionalidades integradas ao sistema como otimizações de performance (*runtime* e bateria), além do *Google Play Protect* e *Play Console*.

A história do Android...

O canal TecMundo apresenta a série de história da tecnologia com a trajetória do Android, o sistema operacional móvel da Google. O vídeo conta a origem e a evolução das versões até chegar na Orion.

Assista o vídeo:
https://www.youtube.com/watch?v=5K4pEk19nhs

1.5.2 Apple iOS

O sistema operacional da Apple é projetado exclusivamente para o hardware produzido por ela. O iOS foi o primeiro a implementar a interface de usuário multitoque e, em grande medida, pode ser visto como responsável pela explosão e universalização no uso de dispositivos móveis. Assim como o sistema operacional para seus computadores de mesa, o MacOS X, o iOS é baseado no *kernel Darwin*, derivado do *FreeBSD*, um sistema livre, similar ao Unix.

Em 2007, a Apple lançou o iPhone, que era considerado por muitos apenas uma versão conectada do iPod. Ainda restrito a algumas operadoras americanas, como a AT&T, não se podia imaginar, na época de seu lançamento, o sucesso que esse dispositivo faria.

O sistema operacional iPhone OS, que, mais tarde, passou a se chamar iOS, aliado a um *hardware* compacto e de alta qualidade, elevou o iPhone a símbolo de status e tornou-o objeto de desejo, fator muito sabiamente trabalhado pelo marketing da Apple.

Não demorou muito para que o iPhone começasse a “roubar” adeptos do *BlackBerry* dentro das corporações, tornando-se o novo “queridinho” dos executivos.

1.5.3 Windows Phone

A Microsoft oferece uma versão de seu sistema operacional, compatível com a API do Windows, mas compilada para o processador ARM. Este sistema operacional não conseguiu ganhar grande popularidade, em claro contraste com o seu domínio na computação tradicional de desktop. O principal fabricante que vende equipamentos com o Windows Phone é a Nokia (que, após ser a empresa líder em telefonia, foi adquirida pela própria Microsoft).

A Microsoft havia desenvolvido o Windows Mobile a partir do Pocket PC, que, como vimos, foi uma evolução do Windows CE. A primeira versão do Windows Mobile, chamada Ozone, foi lançada em junho de 2003 e, a partir dessa versão, diversas outras versões já foram lançadas até hoje.

Por conta da competição de mercado, a Microsoft buscou parceiros, como a Nokia, para a fabricação de equipamentos que utilizassem seu sistema Windows Phone. No entanto, mesmo com a aquisição da Nokia para produção de equipamentos do Windows Mobile (hoje Windows Phone), a Microsoft não obteve uma boa participação de mercado com seu sistema operacional móvel. Em meados de 2016, a Microsoft anunciou que o Windows 10 será a única nomenclatura de sistemas operacionais para os PCs, consoles de jogos e dispositivos móveis, incluindo o *Surface*, misto de tablet e PC.

1.5.4 Symbian

Embora este sistema operacional já tenha sido declarado oficialmente morto, seu efeito no desenvolvimento inicial do segmento foi fundamental e não pode ser ignorado. O Symbian foi a principal plataforma da Nokia em seus dias de glória, assim como para muitos outros fabricantes.Quase todas as empresas que anteriormente vendiam equipamentos com o Symbian mudaram sua oferta para os sistemas Android.

A Ericsson, a Motorola e a Nokia uniram-se para criar o Symbian, sistema operacional adaptado de tecnologias para PDAs que chegou ao mercado em 2001, incorporado a alguns telefones dessas empresas. O Symbian cumpriu seu papel à época, mas ficou obsoleto em menos de uma década, frente aos novos conceitos de sistemas operacionais.

1.5.5 BlackBerryOS

Em 2002, uma empresa canadense de nome RIM (Research in Motion Limited), lançou seu primeiro smartphone, o BlackBerry. Apesar de, em sua primeira versão, o BlackBerry ser um equipamento grandalhão e pouco charmoso, reconhecido de longe por sua cor azul, ele se tornou um padrão global para uso executivo rapidamente.

Com funcionalidades que agradavam em cheio as organizações e conferiam mobilidade aos executivos, o Blackberry dominou o mercado por anos, até o dia em que concorrentes à altura apareceram no horizonte. Assim como a RIM, que criou seus equipamentos movidos por um sistema operacional próprio, o BlackBerryOS, outros fabricantes continuaram a lançar aparelhos e sistemas com a mesma intenção: aparecer frente aos consumidores, que, a essa altura, já tinham dificuldades em entender qual seria o melhor para eles.

Que tal conhecer uma timeline (linha do tempo) completa dos sistemas operacionais com narração?

Essa é a proposta do vídeo elaborado pelo Christian Bruno. descrevendo a linha do tempo dos sistemas operacionais para computadores e dispositivos móveis.

Acesse:
https://www.youtube.com/watch?v=h1CEtMk1CYo

Fica claro, a partir dos sistemas que acabamos de apresentar, bem como da grande maioria dos sistemas operacionais usados para dispositivos móveis, que a diferenciação entre o segmento móvel e a computação tradicional (desktop e servidor) não está no próprio sistema operacional, mas em camadas mais altas. No entanto, a diferença vai muito além de uma mudança na interface do usuário. As características desses dispositivos indubitavelmente determinam questões substantivas e de fundo.

Vejamos algumas das características mais notórias:

a) Armazenamento em estado sólido - A primeira característica notória ao se manusear um telefone ou tablet é que ele não é mais feito com a mesma noção de fragilidade que sempre acompanhou o computador. Os discos rígidos são dispositivos de alta precisão mecânica, e um pequeno acidente pode significar a sua perda absoluta e definitiva. Dispositivos móveis operam com armazenamento em estado sólido, ou seja, em componentes eletrônicos sem partes móveis;

b) Multitarefa, mas monocontexto - A popularização da computação móvel levou a uma forte redução nas expectativas de multitarefa, principalmente porque nos dispositivos móveis há uma carência de memória virtual, e, a memória disponível se torna uma mercadoria escassa. Logo, o sistema operacional é forçado a limitar o número de processos interativos em execução. As interfaces de usuário usadas pelos sistemas móveis abandonam a metáfora da área de trabalho, para voltar ao processo de vários aplicativos em execução, porém com um único programa visível em todos os momentos. Os usuários abrem vários aplicativos, mas normalmente não os finalizam. Se eles não ocuparem toda a memória ficarão abertos para facilitar um novo acesso futuro, de forma mais rápida;

O próprio sistema operacional definirá regras de quando os aplicativos deverão ser finalizados e com qual prioridade a memória será liberada;

c) Consumo elétrico - A economia do consumo elétrico tem duas vertentes principais: por um lado, o desenvolvimento de *hardware* mais energeticamente eficiente, independentemente da maneira em que opera e, por outro, a criação de mecanismos por meio dos quais um equipamento de informática pode detectar mudanças no padrão de atividade (ou o operador pode indicar uma mudança na resposta esperada), e este último reage reduzindo sua demanda (que é tipicamente obtida pela redução da velocidade de certos componentes do equipamento). A economia de energia que esses padrões de uso proporcionam não se deve apenas ao *hardware* cada vez mais eficiente usado pelos dispositivos móveis, mas também à programação de aplicativos nos quais os desenvolvedores procuram explicitamente padrões eficientes, suspensão fácil, e minimização na necessidade de acordar o *hardware*;

d) Ambiente em mudança - Nos *desktops* e servidores as mudanças de ambiente não são tão comuns. Normalmente não há grandes mudanças de rede, configurações ou consumo de energia. Uma das mudanças mais complexas de se implementar em sistemas operacionais de dispositivos móveis foi a de fornecer a plasticidade necessária nestes diferentes aspectos: o dispositivo móvel deve ser mais enérgico (consumir mais) em suas mudanças no perfil de energia, respondendo a um ambiente em mudança. Você pode aumentar ou diminuir o brilho da tela, dependendo do brilho ao redor, ou desabilitar determinadas funcionalidades se ela já estiver em níveis críticos de carga.

Com relação à rede, por exemplo, você deve ser capaz de aproveitar as conexões fugazes enquanto o usuário está em movimento, iniciando eventos como a sincronização. Finalmente, é claro, a interface do usuário: os dispositivos móveis não têm uma orientação natural # exclusiva, como fazem os computadores. As interfaces do usuário devem ser projetadas para que possam ser reconfiguradas agilmente antes da rotação da tela;

e) O jardim murado - Uma consequência indireta (e não técnica) do nascimento de plataformas móveis é a popularização de um modelo de distribuição de *software* conhecido como jardim murado (plataforma fechada). A Apple, a Microsoft e a Google adotaram esse modelo de distribuição de aplicativos. A Apple com a Apple Store, a Microsoft com o Windows Phone Store e a Google com o *Google Play*. A peculiaridade deste modelo é que qualquer desenvolvedor pode criar um aplicativo e enviá-lo, porém as fabricantes se reservam no direito de aprová-lo, ou excluí-lo a qualquer momento. Ou seja, esse modelo permite que elas se tornem uma espécie de juiz, que pode determinar o que um usuário pode ou não instalar e executar.

Bora rever!

Essa unidade apresentou uma introdução sobre os Sistemas Operacionais para computadores e dispositivos móveis. Foi possível perceber a importância de se conhecer o surgimento e evolução do sistema operacional, particularmente para dispositivos móveis.

Os sistemas operacionais estão presentes nos mais diversos dispositivos, desde computadores pessoais até dispositivos móveis. O sistema operacional atua como uma camada de *software* entre os programas aplicativos dos usuários finais e o *hardware*. Ele é uma estrutura complexa de *software*, bastante ampla, que incorpora aspectos de alto nível, como a interface gráfica e os programas utilitários, e de baixo nível, como a gerência de memória e os drivers de dispositivos.

Foi possível observar que o sistema operacional atua como uma camada transparente para o usuário permitindo que o computador seja utilizado de forma conveniente e eficiente, ocultando a complexidade do *hardware*. O sistema operacional como gerente de recursos faz a gestão do uso do processador, do espaço em memória, do acesso à arquivos, conexões de rede e dispositivos externos, sempre buscando evitar conflitos.

A diferenciação entre o segmento móvel e a computação tradicional (desktop e servidor) não está no próprio sistema operacional, mas em camadas mais altas. A diferença vai muito além de uma mudança na interface do usuário e passa por características marcantes como armazenamento e consumo de energia.

Na próxima unidade vamos entender o funcionamento dos sistemas operacionais e explorar mais a sua arquitetura.

Glossário

***Batch*:** lote, conjunto, agregar as coisas em conjuntos;

***Desktop*:** parte da interface gráfica de sistemas operacionais que exibe, no vídeo, representações de objetos usualmente presentes nas mesas de trabalho, como documentos, arquivos, pastas e impressoras; área de trabalho;

***Hackers*:** em informática, *hacker* é um indivíduo que se dedica, com intensidade incomum, a conhecer e modificar os aspectos mais internos de dispositivos, programas e redes de computadores;

***Hardware*:** conjunto dos componentes físicos (material eletrônico, placas, monitor, equipamentos periféricos etc.) de um computador;

Interface gráfica: é um conceito da forma de interação entre o usuário do computador e um programa por meio de uma tela ou representação gráfica, visual, com desenhos, imagens, etc;

***Job*:** trabalho ou tarefa;

Linguagem de máquina: um programa em código de máquina consiste de uma sequência de bytes que se tratam de instruções a serem executadas pelo processador;

***Mobile*:** que se move; móvel, móbil;

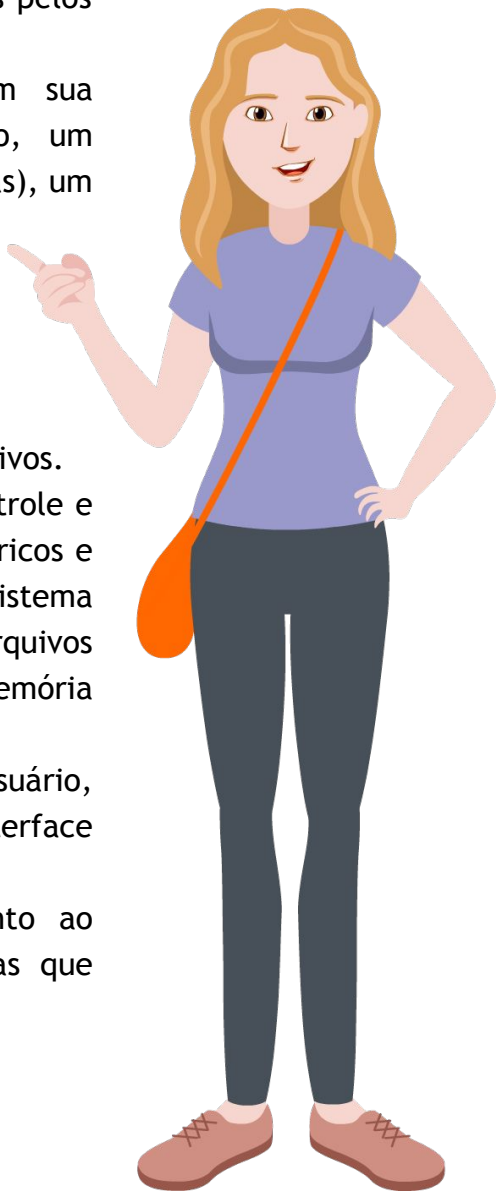
***Timesharing*:** tempo compartilhado;

***Worms*:** (termo da língua inglesa que significa, literalmente, "verme") é um programa autorreplicante, diferente de um vírus.

ENTENDENDO OS SISTEMAS OPERACIONAIS

Como vimos, o sistema operacional é uma camada de *software* que atua entre o *hardware* e os programas aplicativos, utilizados pelos usuários finais.

Todo sistema operacional possui aspectos básicos em sua organização sendo constituído normalmente por: um núcleo, um gerenciador de memória, um gerenciador de E/S (entradas e saídas), um sistema de arquivos e um processador de comandos/interface com o usuário.



O núcleo é responsável pela gerência do processador, tratamento de interrupções, comunicação e sincronização entre processos, enquanto o gerenciador de memória é responsável pelo controle e alocação de memória aos processos ativos.

O gerenciador de entradas e saídas é responsável pelo controle e execução de operações de E/S, pela otimização do uso de periféricos e pela interface de comunicação com o usuário. Dentro do sistema operacional, para manipular informações, existe um sistema de arquivos encarregado pelo acesso e integridade dos dados residentes na memória secundária (discos e dispositivos de armazenamento).

O processador de comandos atua como interface com o usuário, assim como o processador de E/S, também, é responsável pela interface de comunicação com o usuário.

Os sistemas operacionais podem ser classificados quanto ao número de usuários e, também, quanto ao número de tarefas que podem executar. Vejamos:

Quanto ao número de usuários:

- Monousuário:
  - Projetados para suportar um único usuário.
  - Ex: MS-DOS, *Windows* 3x, *Windows* 9x.

- Multiusuário:
  - Projetados para suportar várias sessões de usuários.
  - Ex: *Windows* NT(2000), UNIX.

Quanto ao número de tarefas:

- Monotarefa:
  - Capazes de executarem apenas uma tarefa (um aplicativo) de cada vez;
  - Os recursos computacionais estão inteiramente dedicados a um único programa/tarefa;
  - A UCP fica ociosa durante muito tempo enquanto o programa aguarda por um evento (digitação de um dado, leitura do disco, etc.);
  - A memória principal é subutilizada caso o programa não a preencha totalmente;
  - Os periféricos são dedicados a um único usuário;
  - Não existem grandes preocupações com a proteção de memória;
  - A complexidade de implementação é relativamente baixa. Ex: MS-DOS

- Multitarefa:
  - Capazes de executarem várias atividades simultaneamente, como uma compilação e um processamento de texto;
  - Vários programas competem pelos recursos do sistema;
  - O objetivo é manter mais de um programa em execução "simultaneamente", dando a ilusão de que cada programa/usuário tem a máquina dedicada para si;
  - A ideia é tirar proveito do tempo ocioso da UCP durante as operações de E/S. Enquanto um programa espera por uma operação de leitura ou escrita, os outros podem serem processados no mesmo intervalo;
    - Maximização do uso do processador e da memória;
    - Maior taxa de utilização do sistema como um todo (redução do custo total máquina/homem);
  - Suporte de *hardware*:
    - Proteção de memória;
    - Mecanismo de interrupção (sinalização de eventos);
    - Discos magnéticos (acesso randômico aos programas, melhor desempenho em operações de E/S) para implementação de memória virtual;

  Ex: *Windows*, OS/2, Unix.

2.1 Abstração e Gerência de Recursos

Dentro dos sistemas computacionais, cada hardware tem suas peculiaridades e cabe ao sistema operacional gerenciar essas diferenças de forma transparente. Por exemplo, um processador de textos (software) não necessita saber como ocorre o processo de acesso e gravação de um arquivo (hardware). Ele não deve se preocupar em como os dispositivos são acessados.

Compete ao sistema operacional prover interfaces de acesso aos dispositivos, facilitando aos softwares formas mais simples de usar esses recursos que as interfaces de baixo nível. Neste processo o sistema operacional tornará os aplicativos independentes do hardware, ou seja, ele irá definir interfaces de acesso homogêneas (padronizadas) para dispositivos com tecnologias distintas (diferentes).

O sistema operacional também é o responsável pela definição das políticas para o gerenciamento do uso dos recursos de hardware pelos aplicativos (softwares), efetuando a resolução de possíveis disputas e conflitos que possam ocorrer. Imagine que dois softwares querem, por exemplo, acessar um arquivo ao mesmo tempo, em um único dispositivo de leitura. Será o sistema operacional que fará a mediação do acesso e uso do processador, do disco e da memória, por exemplo.

2.2 Tipos de sistemas operacionais

a) *Batch*

Nos sistemas operacionais do tipo *batch* todos os programas para execução são colocados em uma fila. Assim, o processador recebia um programa após o outro, realizando o processamento em sequência, ampliando o grau (capacidade) de utilização do sistema. Atualmente, os sistemas operacionais em *batch* são pouco utilizados, porém o termo "lote" é empregado até hoje. Esse termo ainda é utilizado para definir um conjunto de comandos que são executados sem a interferência do usuário.

b) Rede

Os sistemas operacionais em rede tem suporte nativo para a operação em rede e é a maioria dos sistemas operacionais atuais. Podem ser utilizados para o compartilhamento de recursos de vários computadores e, também, podem compartilhar os seus próprios recursos. Atuam de forma independente, e caso a conexão entre um dos nós sofra qualquer problema, os demais continuam operando normalmente. É claro que com a desconexão de um nós alguns recursos poderão se tornar indisponíveis, mas isso não irá interferir no funcionamento dos demais recursos.

c) Distribuído

O sistema operacional distribuído apresenta a característica principal de que os recursos de cada máquina estão disponíveis globalmente, de forma transparente aos usuários. Assim, o sistema operacional distribuído se apresenta para o usuário e suas aplicações como um único sistema centralizado. Para o usuário e aplicações é como se não existisse uma rede de computadores, ou seja, o usuário não percebe qual computador da rede está utilizando. Esse tipo de sistema operacional ainda não é uma realidade de mercado. Como exemplo podemos citar o sistema operacional Amoeba[1], que pode ser executado em diversas plataformas, incluindo SPARC, i386, 68030, Sun 3/50 e Sun 3/60, utilizando o *FLIP* (Fast-Local-Internet-Protocol) como protocolo de rede. Trata-se de uma suíte de protocolos *Internet* que provê transparência, segurança e gerenciamento de rede. É nesse sistema operacional que a linguagem de programação *Python* foi originalmente desenvolvida.

d) Multiusuário

A maioria dos sistemas operacionais atuais é considerada multiusuário, pois permite a utilização por múltiplos usuários simultâneos. Esse tipo de sistema operacional deve suportar a identificação do "dono" de cada recurso dentro do sistema, ou seja, os recursos como arquivos, processos e até as conexões de rede devem ser executados com a identificação inequívoca a qual o usuário pertence. Naturalmente, por questões de segurança, esse tipo de sistema operacional irá implementar (impor) regras de controle de acesso para impedir o uso desses
recursos por usuários não autorizados.

e) Desktop

Os sistemas operacionais para *desktop* são conhecidos como sistema operacional "de mesa", pois normalmente são aplicados em computadores pessoais (usuários domésticos), portáteis (*notebooks*) ou *desktops* comuns (corporativos). São sistemas operacionais que dão suporte a atividades comuns, tradicionais e corriqueiras. Possuem ambiente gráfico (interface) amigável, com suporte a rede (conectividade) e grande nível de interatividade.

f) Servidor

Os sistemas operacionais para servidor trabalham com a gestão de grandes quantidades de recursos, como discos, memórias e processadores. São preparados para trabalhar com múltiplos usuários conectados ao mesmo tempo, até mesmo por diferentes meios (conexões remotas e locais). Neste tipo de sistema operacional o suporte a rede é nativo, ou seja, é parte integrante do sistema operacional não havendo versão sem esse tipo de suporte.

Atualmente, é uma prática comum de mercado desenvolver sistemas operacionais que possuem versões para *Desktop* e para Servidores (Ex: *Ubuntu* Server ou *Windows Server*).

g) Embutido

Os sistemas operacionais embutidos são aqueles instalados em *hardwares* com pouca capacidade de processamento (adaptados), como celulares (*smartphones*), câmeras, GPS, calculadoras, tocadores de MP3, PDAs, *tablets* e outros pequenos dispositivos. Normalmente são aplicados em *hardwares* que possuem funções específicas, tendo memória limitada, processador mais lento e *display* de pequenas dimensões. Neste caso, tanto o sistema operacional e as aplicações são projetados para minimizar o uso do processador (redução do consumo da bateria). Podem fazer uso de tecnologias *wireless*, como *Bluetooth* e *Wi-fi*, para acesso remoto a *e-mail* e navegação *Web*. Porém, é importante ressaltar que alguns dispositivos móveis e *hardwares* para ambientes de testes e simulações cresceram tanto em capacidade de memória e processamento que estão utilizando versões de sistemas operacionais semelhantes ou até iguais aos utilizados em *Desktop*, como, por exemplo, os celulares (*smartphones*).

h) Tempo real

Os Sistemas Operacionais de Tempo Real são sistemas utilizados quando há necessidade de um comportamento temporal previsível, ou seja, é importante considerar o tempo (parâmetro fundamental) de resposta no melhor caso e pior caso de operação.

Essa categoria de sistema operacional possui dois tipos:

- *soft real-time systems (ou sistema de tempo real não crítico)* - neste tipo de sistema operacional de tempo real o descumprimento do prazo é aceitável e não causa dano permanente. Temos como exemplo os sistemas de áudio digital, multimídia e telefones digitais. Nestes casos, a perda de prazo implica em degradação do serviço prestado (gravação de CD);
- *hard real-time systems (ou sistema de tempo real crítico)* - neste outro tipo de sistema operacional de tempo real a perda de prazo pode causar grandes prejuízos econômicos ou ambientais (usina nuclear, caldeiras industriais). Essa categoria de sistema operacional deve fornecer garantia absoluta de que determinada ação ocorrerá em determinado momento.

2.3 Funcionalidades e conceitos de *hardware*

Além das funcionalidades básicas oferecidas pela maioria dos sistemas operacionais, várias outras vêm se agregar aos sistemas modernos, para cobrir aspectos complementares, como a interface gráfica, suporte de rede, fluxos multimídia, gerência de energia, etc.

As funcionalidades do sistema operacional geralmente são interdependentes: por exemplo, a gerência do processador depende de aspectos da gerência de memória, assim como a gerência de memória depende da gerência de dispositivos e da gerência de proteção. Alguns autores [Silberschatz et al., 2001, Tanenbaum, 2003] representam a estrutura do sistema operacional conforme indicado na Figura 1. Nela, o núcleo central implementa o acesso de baixo nível ao *hardware*, enquanto os módulos externos representam as várias funcionalidades do sistema.

Um sistema operacional possui diversas funcionalidades e deve atuar em muitas frentes. O sistema possui inúmeros recursos que devem ser gerenciados, como por exemplo, processador, memória, dispositivos físicos, arquivos e proteção.

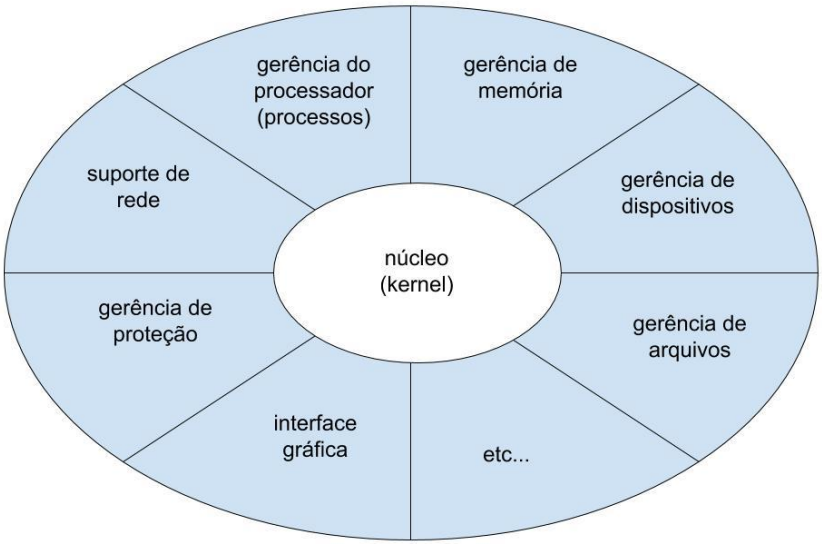


Figura 1 - Os módulos de gerência do sistema operacional são interdependentes.
Fonte: : [Silberschatz et al., 2001, Tanenbaum, 2003]

Uma regra importante a ser observada na construção de um sistema operacional é a separação entre os conceitos de política e mecanismo. Como política consideram-se os aspectos de decisão mais abstratos, que podem ser resolvidos por algoritmos de nível mais alto, como por exemplo, decidir a quantidade de memória que cada aplicação ativa deve receber, ou qual o próximo pacote de rede a enviar para satisfazer determinadas especificações de qualidade de serviço.

Por outro lado, como mecanismo consideram-se os procedimentos de baixo nível usados para implementar as políticas, ou seja, atribuir ou retirar memória de uma aplicação, enviar ou receber um pacote de rede, etc. Os mecanismos devem ser suficientemente genéricos para suportar mudanças de política sem necessidade de modificações. Essa separação entre os conceitos de política e mecanismo traz uma grande flexibilidade aos sistemas operacionais, permitindo alterar sua personalidade (sistemas mais interativos ou mais eficientes) sem ter de alterar o código que interage diretamente com o *hardware*.

O sistema operacional interage diretamente com o *hardware* disponibilizando serviços às aplicações. Ainda hoje, a grande maioria dos computadores com apenas um processador segue a arquitetura básica definida por János (John) Von Neumann, a cerca de 40 anos, intitulada como “arquitetura Von Neumann”. A principal característica desse modelo é que o programa a ser executado reside na memória junto com os dados, conhecida como “programa armazenado”.

É por meio de um ou mais barramentos (para a transferência de dados, endereços e sinais de controle) que os principais elementos constituintes do computador estão interligados.

A composição normal de um sistema computacional típico é possuir um ou mais processadores com a responsabilidade de executar as instruções das aplicações, armazenadas em uma área de memória que contém as aplicações em execução (seus códigos e dados) assim como dispositivos periféricos que permitem o armazenamento de dados e a comunicação com o mundo exterior, como discos rígidos, terminais e teclados.

A parte central de um sistema computacional é o processador o qual tem a função de continuamente ler instruções e dados da memória ou de periféricos, fazer o processamento e encaminhar novamente o resultado para a memória ou a outros dispositivos (periféricos). O processador contém a unidade lógica aritmética (ULA) para fazer operações lógicas e cálculos, alguns registradores especiais (ponteiro de pilha, contador de programa, *flags* de status, etc.) e um grupo de registradores para armazenar dados de trabalho.

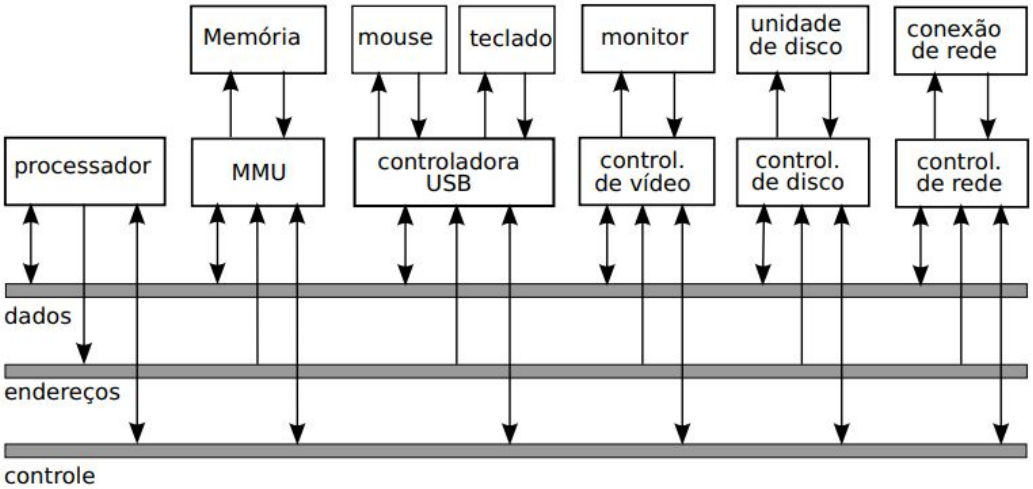


Figura 2 - Arquitetura de um computador típico.
Fonte: Maziero (2017, p.10)

Os barramentos são responsáveis por realizar todas as transferências de dados entre o processador, a memória e os periféricos. Temos:

- o barramento de endereços indica a posição de memória (ou o dispositivo) a acessar;
- o barramento de controle indica a operação a efetuar (leitura ou escrita); e
- o barramento de dados transporta a informação indicada entre o processador e a memória ou um controlador de dispositivo.

Normalmente, a mediação de acesso à memória é realizada por um controlador específico (que pode estar fisicamente dentro do próprio processador) chamado Unidade de Gerência de Memória (MMU - *Memory Management Unit*). Sua função é analisar cada endereço solicitado pelo processador, validá-los, efetuar as conversões de endereçamento necessárias e executar a operação solicitada pelo processador (leitura ou escrita de uma posição de memória).

Os controladores são circuitos específicos que permitem o acesso aos periféricos do computador (discos, teclado, monitor, etc.). Assim, o controlador USB permite acesso ao *mouse*, teclado e outros dispositivos USB externos, a placa de vídeo permite o acesso ao monitor e a placa *ethernet* dá acesso à rede.

O processador vê cada dispositivo representado por seu respectivo controlador. As portas de entrada/saída endereçáveis dão acesso aos controladores. Assim, a cada controlador é atribuída uma faixa de endereços de portas de entrada/saída.

Temos duas alternativas de comunicação quando um controlador de periférico tem uma informação importante a fornecer ao processador:

- Informar o processador através do barramento de controle, enviando a ele uma requisição de interrupção (IRQ – *Interrupt ReQuest*);
- Esperar até que o processador o consulte, o que poderá ser demorado caso o processador esteja ocupado com outras tarefas (o que geralmente ocorre).

Quando os circuitos do processador recebem uma requisição de interrupção, eles suspendem seu fluxo de execução corrente e desviam para um endereço pré-definido, onde está uma rotina de tratamento de interrupção (*interrupt handler*). A rotina é responsável por tratar a interrupção, ou seja, executar as ações necessárias para atender ao dispositivo que a gerou. Ao final da rotina de tratamento da interrupção, o processador retoma o código que estava executando quando recebeu a requisição.

É função do sistema operacional gerenciar os recursos do *hardware*, e fornecê-los às aplicações de acordo com as suas necessidades. Para garantir a integridade dessa gerência, é primordial garantir que as aplicações não consigam acessar o *hardware* diretamente. Esse acesso só deve ocorrer por meio de pedidos ao sistema operacional, que irá avaliar e intermediar todos os acessos ao *hardware*.

Mas como impedir que o *hardware* seja acessado diretamente pelas aplicações?

Para permitir diferenciar os privilégios de execução dos diferentes tipos de *software*, os processadores modernos contam com dois ou mais níveis de privilégio de execução. Esses níveis são controlados por *flags* especiais nos processadores, e as formas de mudança de um nível de execução para outro são controladas estritamente pelo processador.

Podemos considerar dois níveis básicos de privilégio:

- Nível núcleo: também denominado nível supervisor, sistema, monitor ou ainda *kernel space*. Para um código executando nesse nível, todo o processador está acessível: todos os recursos internos do processador (registradores e portas de entrada/saída) e áreas de memória podem ser acessados. Além disso, todas as instruções do processador podem ser executadas. Ao ser ligado, o processador entra em operação neste nível.

- Nível usuário (ou *userspace*): neste nível, somente um subconjunto das instruções do processador, registradores e portas de entrada/saída estão disponíveis. Instruções "perigosas" como *HALT* (parar o processador) e *RESET* (reiniciar o processador) são proibidas para todo código executando neste nível. Além disso, o *hardware* restringe o uso da memória, permitindo o acesso somente a áreas previamente definidas. Caso o código em execução tente executar uma instrução proibida ou acessar uma área de memória inacessível, o *hardware* irá gerar uma exceção, desviando a execução para uma rotina de tratamento dentro do núcleo, que provavelmente irá abortar o programa em execução (e também gerar a famosa frase "este programa executou uma instrução ilegal e será finalizado", no caso do *Windows*).

É fácil perceber que, em um sistema operacional convencional, o núcleo e os drivers operam no nível núcleo, enquanto os utilitários e as aplicações operam no nível usuário, confinados em áreas de memória distintas, conforme ilustrado na Figura 3.

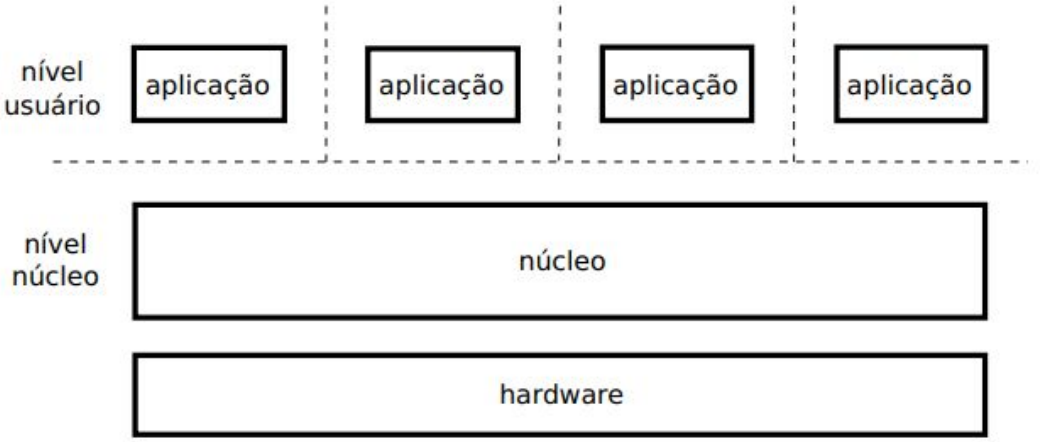


Figura 3 - Separação entre o núcleo e as aplicações
Fonte: Maziero (2017, p.15)

Os mapeamentos de memória realizados pela MMU nos acessos em nível usuário impõem o confinamento de cada aplicação em sua área de memória, provendo robustez e confiabilidade ao sistema, já que garante que uma aplicação não poderá interferir nas áreas de memória de outras aplicações ou do núcleo.

Porém, essa proteção introduz um novo problema: como acessar, a partir de uma aplicação, as rotinas oferecidas pelo *kernel* (núcleo) para o acesso ao *hardware* e suas abstrações?

Em outras palavras, como uma aplicação sem ter privilégio, para acessar as portas de entrada/saída correspondentes e sem poder invocar o código do núcleo que implementa esse acesso (pois esse código reside em outra área de memória), poderá acessar a placa de rede para enviar/receber dados?

A solução para esse contratempo está no mecanismo de interrupção. Uma instrução especial, implementada pelos processadores, permite acionar o mecanismo de interrupção de forma intencional, sem depender de eventos externos ou internos.

As chamadas de sistema (*system call ou syscall*) são assim denominadas porque são realizadas pela ativação de procedimentos do núcleo usando interrupções de *software* (ou outros mecanismos correlatos).

Todas as operações envolvendo abstrações lógicas (criação e finalização de tarefas, operadores de sincronização e comunicação, etc.) ou o acesso a recursos de baixo nível (periféricos, arquivos, alocação de memória, etc.) são definidas por chamadas de sistema pelos sistemas operacionais.

É por meio de uma biblioteca do sistema (*system library*), que prepara os parâmetros, invoca a interrupção de software e retorna à aplicação os resultados obtidos, que normalmente as chamadas de sistema são oferecidas para as aplicações em modo usuário.

Centenas de chamadas de sistema distintas, para as mais diversas finalidades são implementadas pela maioria dos sistemas operacionais.

A API (*Application Programming Interface*) de um sistema operacional é definida pelo conjunto de chamadas de sistema oferecidas por seu núcleo.

A Win32 é um exemplo de API bem conhecida. Ela é oferecida pelos sistemas *Microsoft* derivados do *Windows* NT. Outro exemplo é a API POSIX que define um padrão de interface de núcleo para sistemas UNIX.

2.4 Estrutura de um sistema operacional

Não podemos considerar o sistema operacional como um bloco de *software* fechado e único rodando sobre o *hardware*. Sua composição real vai, além disso. Ele é composto por diversos componentes, em que cada um possui um objetivo e uma funcionalidade complementar. A tabela a seguir apresenta os componentes mais importantes de um sistema operacional típico.

Núcleo	É a parte central (coração) do sistema operacional responsável por gerenciar os recursos do *hardware* que são utilizados pelas aplicações. Implementa as principais abstrações que os programas aplicativos utilizam.
Drivers	São módulos específicos de código que permitem acessar determinados dispositivos físicos. Cada dispositivo (discos rígidos IDE, SCSI, portas USB, placas de vídeo, etc.) necessita de um *driver* específico. O *driver* normalmente é construído pelo próprio fabricante do *hardware*. É disponibilizado em linguagem de máquina (de forma compilada) para ser acoplado ao sistema operacional permitindo o uso do dispositivo físico.
Código de inicialização	É responsável por realizar o carregamento do núcleo do sistema operacional em memória e começar a sua execução. Para inicializar o *hardware* necessitamos de um conjunto de tarefas bastante complexas, como reconhecer os dispositivos instalados, configurá-los e testá-los de forma correta para seu uso futuro.
Programas utilitários	São aplicativos (programas) que permitem o uso simples e fácil do sistema operacional, disponibilizando funcionalidades complementares ao núcleo (*kernel*) permitindo a formatação de discos e mídias, manipulação de arquivos (criar, mover, remover), configuração de novos dispositivos, terminal, interpretador de comandos, interface gráfica, gerenciador de janelas, entre outros.

A Figura 4 apresenta a inter relação entre as diversas partes do sistema operacional. É importante considerar que o relacionamento e a interligação entre esses componentes podem variar de acordo com o sistema operacional, conforme veremos no tópico 2.5 - Arquiteturas de sistemas operacionais.

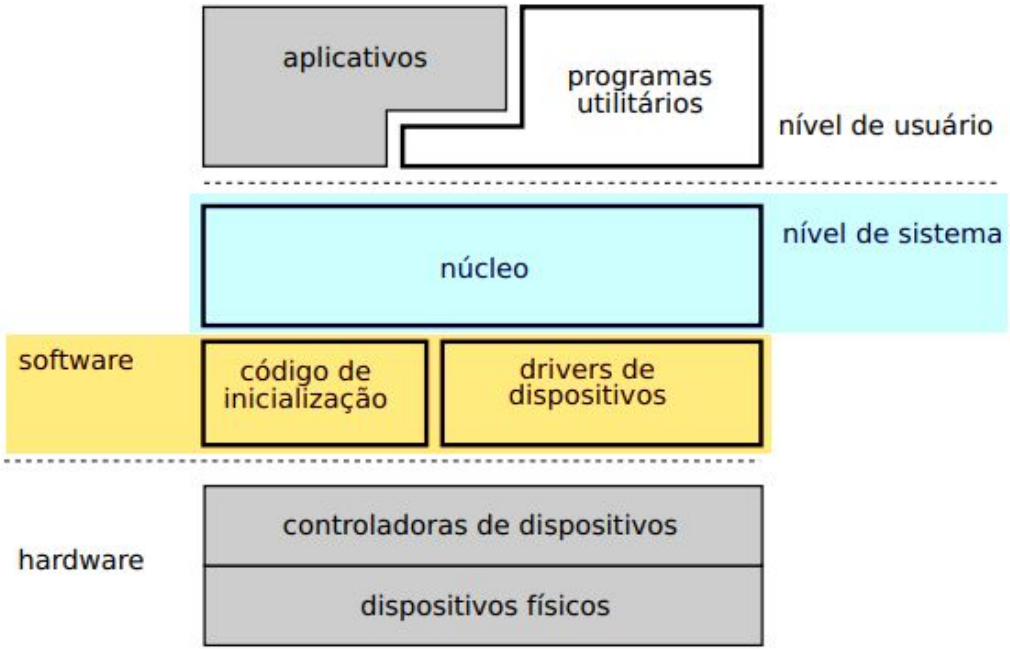


Figura 4 - Estrutura de um sistema operacional
Fonte: Adaptado de Maziero (2017, p.9)

2.5 Arquiteturas de sistemas operacionais

As várias partes que compõem o sistema podem ser organizadas de diversas formas. Podem-se separar as suas funcionalidades e modularizar o seu projeto, mesmo que a definição de níveis de privilégio imponha uma estruturação mínima a um sistema operacional.

Vejamos agora as arquiteturas mais populares para a organização de sistemas operacionais:

a) Sistemas monolíticos

Quando todos os componentes do núcleo operam em modo de núcleo e se inter-relacionam conforme suas necessidades, sem restrições de acesso entre si, pois o código no nível núcleo tem acesso pleno a todos os recursos e áreas de memória, estamos atuando em um sistema operacional monolítico.

Em um sistema operacional monolítico não há barreiras impedindo acessos, e qualquer componente do núcleo pode acessar os demais componentes, toda a memória ou mesmo dispositivos periféricos diretamente. A grande vantagem dessa arquitetura é seu desempenho. Como resultados da interação direta entre componentes têm sistemas mais compactos.

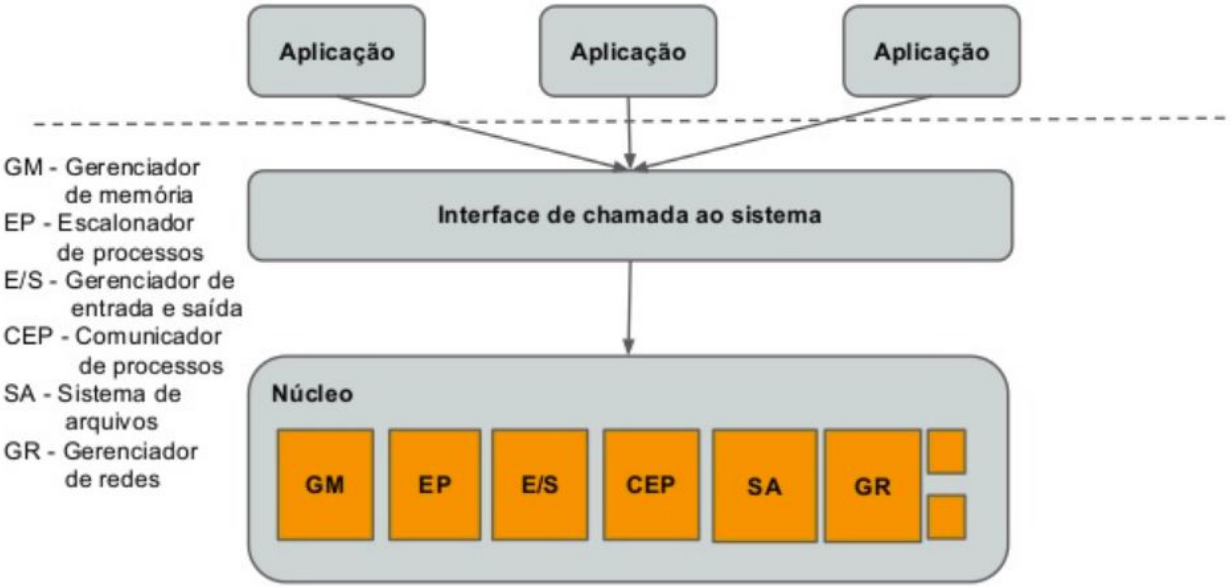


Figura 5 - Uma arquitetura monolítica.
Fonte: elaborado pelo autor.

Os primeiros sistemas operacionais foram organizados utilizando a arquitetura monolítica. Temos como exemplos, os sistemas UNIX antigos e o MS-DOS. Devido às limitações do *hardware* sobre o qual executam apenas os sistemas operacionais embarcados continuam atualmente empregando arquitetura monolítica. Mesmo o núcleo do GNU/*Linux*, que nasceu monolítico, mas vem sendo modificado e modularizado embora uma boa parte de seu código ainda permaneça no nível de núcleo.

b) Sistemas em camadas

Podemos utilizar camadas para estrutura de forma mais elegante um sistema operacional. Enquanto a camada superior define a interface do núcleo para as aplicações (as chamadas de sistema), as camadas intermediárias provêm níveis de abstração e gerência cada vez mais sofisticados. Por fim, a camada mais baixa realiza a interface com o *hardware*.

O modelo de referência OSI (*Open Systems Interconnection*) implementou essa abordagem de estruturação de *software* fazendo muito sucesso no domínio das redes de computadores, logo seria algo natural esperar sua adoção no domínio dos sistemas operacionais.

Porém, não foi bem assim, pois alguns inconvenientes limitam sua aceitação nesse contexto. Vejamos:

- O pedido de uma aplicação demora mais tempo para chegar até o dispositivo periférico ou recurso a ser acessado, devido ao empilhamento de várias camadas de *software* prejudicando o desempenho do sistema;
- Não é óbvio como dividir as funcionalidades de um núcleo de sistema operacional em camadas horizontais de abstração crescente, pois essas funcionalidades são interdependentes, embora tratem muitas vezes de recursos distintos.

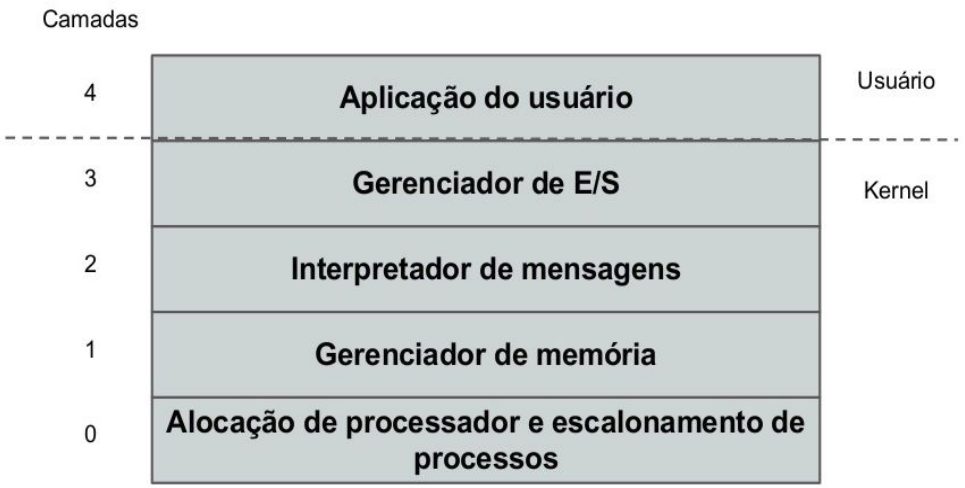


Figura 6 - Uma arquitetura em camadas.
Fonte: elaborado pelo autor.

A estruturação em camadas é adotada apenas parcialmente hoje em dia, em decorrência desses inconvenientes. Assim como o *Windows* NT e seus sucessores que utilizavam a camada HAL – *Hardware Abstraction Layer*, muitos sistemas implementam uma camada inferior de abstração do *hardware* para interagir com os dispositivos. Como exemplos de sistemas fortemente estruturados em camadas temos o IBM OS/2 e o MULTICS. Esses sistemas operacionais também organizam em camadas alguns subsistemas como a gerência de arquivos e o suporte de rede (seguindo o modelo OSI).

c) Sistemas micronúcleo (microkernel)

Essa abordagem torna os núcleos de sistema menores. Por isso ela foi chamada de micronúcleo (ou µ-kernel). Os sistemas operacionais *microkernel* retiraram do núcleo todo o código de alto nível (normalmente associado às políticas de gerência de recursos). É outra possibilidade de estruturação em oposição aos sistemas monolíticos. Os sistemas operacionais micronúcleo deixam dentro do núcleo somente o código de baixo nível necessário para interagir com o *hardware* e criar as abstrações fundamentais (como a noção de atividade).

Nesse tipo de abordagem, *microkernel*, o código de acesso aos blocos de um disco rígido seria mantido no núcleo. Já as abstrações de arquivo e diretório seriam criadas e mantidas por um código fora do núcleo, executando da mesma forma que uma aplicação do usuário.

Somente a noção de atividade, de espaços de memória protegidos e de comunicação entre atividades é implementada normalmente em um micronúcleo.

Políticas de uso do processador e da memória, o sistema de arquivos e o controle de acesso aos recursos, considerados aspectos de alto nível, são implementados fora do núcleo, em processos que se comunicam usando as primitivas do núcleo. A Figura 7 ilustra essa abordagem.

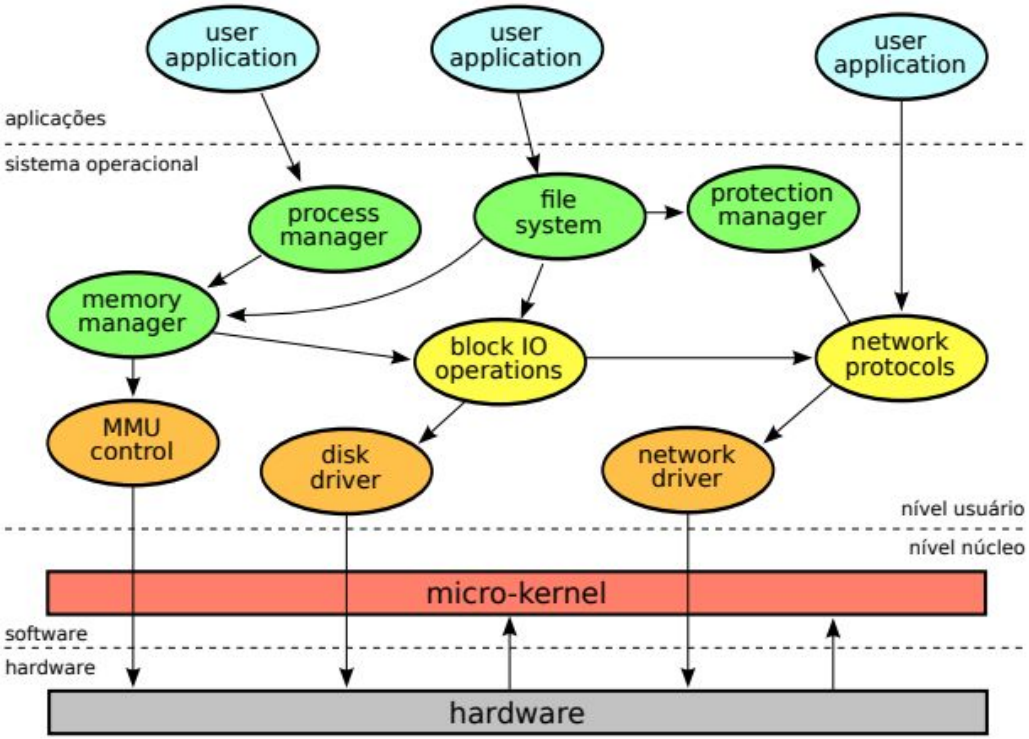


Figura 7 - Visão geral de uma arquitetura micronúcleo.
Fonte: Maziero (2017, p.20)

As interações entre componentes e aplicações, em um sistema micronúcleo, são feitas através de trocas de mensagens. Desta forma, se uma aplicação deseja acessar (abrir) um arquivo no disco rígido, envia uma mensagem para o gerente de arquivos que se comunica com o gerente de dispositivos para obter os blocos de dados relativos ao arquivo desejado.

Devido às restrições impostas pelos mecanismos de proteção do *hardware* os processos não podem se comunicar diretamente. Dessa maneira, todas as mensagens são transmitidas através de serviços do micronúcleo, como mostra a Figura 7.

Essa abordagem também foi chamada cliente/servidor, pois os processos têm de solicitar "serviços" uns dos outros, para poder realizar suas tarefas.

Utilização real do *microkernel*

Durante os anos 80 os micronúcleos foram muito investigados e as principais vantagens são sua robustez e flexibilidade. Os sistemas Mach e Chorus são dois exemplos clássicos de sistemas operacionais *microkernel*. O sistema MacOS X da Apple e o Digital UNIX tem suas raízes no sistema Mach, e adotam parcialmente essa estruturação (*microkernel*). O QNX empregado em sistemas embarcados e de tempo real é um dos poucos exemplos de micronúcleo amplamente utilizado.

Se um subsistema do sistema operacional micronúcleo tiver problemas, os níveis de privilégio e os mecanismos de proteção de memória irão deter o erro, impedindo que o restante do sistema seja afetado pela instabilidade.

Outro aspecto interessante é a possibilidade de customização do sistema operacional, pois é possível iniciar somente os componentes necessários ou mais adequados às aplicações que serão executadas.

Uma desvantagem desse tipo de sistema operacional é o desempenho que fica prejudicado devido ao custo associado às trocas de mensagens entre componentes, diminuindo assim a aceitação desta abordagem.

d) Máquinas virtuais

A execução de programas e bibliotecas em uma determinada plataforma computacional depende sempre da compilação para essa plataforma específica, respeitando o conjunto de chamadas do sistema operacional e o conjunto de instruções do processador.

A analogia de execução de um sistema operacional sobre uma plataforma de *hardware* é a mesma. Ele só poderá ser executado sobre a plataforma se for compatível com ela.

Assim, é importante entender que uma biblioteca só funciona sobre o *hardware* e o sistema operacional para os quais foi projetada. Da mesma forma, as aplicações também têm de obedecer a interfaces pré-definidas. Assim, de forma geral, um sistema operacional só funciona sobre o *hardware* para o qual foi construído.

Quando observamos os sistemas operacionais atuais percebemos que as interfaces de baixo nível são pouco flexíveis. Existem proteções e geralmente não é possível criar novas instruções de processador ou novas chamadas de sistema, ou ainda mudar sua semântica.

Mas como executar um Sistema Operacional sobre um *hardware* para o qual ele não foi criado?

Para resolver essa questão **podemos utilizar técnicas de virtualização** e contornar os problemas de compatibilidade entre os componentes de um sistema.

E o que é Virtualização afinal?
Virtualização (em computação) é a criação de uma versão virtual de alguma coisa, como um sistema operacional, um servidor, um dispositivo de armazenamento (*storage*) ou recurso de rede.

Vamos saber mais?
Acesse:
http://www.jvasconcellos.com.br/unijorge/wp-content/uploads/2012/01/cap4-v2.pdf

Podemos criar uma camada intermediária de *software* que disponibilize aos demais componentes serviços com outra interface. Isso é possível usando os serviços oferecidos por um determinado componente do sistema.

O sistema computacional visto através dessa camada é denominado máquina virtual, uma vez que por meio dessa camada de compatibilidade podemos acoplar interfaces distintas permitindo que um programa desenvolvido para uma plataforma A possa ser executado sobre uma plataforma distinta B.

Um exemplo de máquina virtual pode ser observado na Figura 8. Neste exemplo temos uma camada de virtualização sobre uma plataforma de *hardware Sparc* permitindo a execução de um sistema operacional *Windows* e suas aplicações, distinta daquela para a qual foi projetado (Intel/AMD).

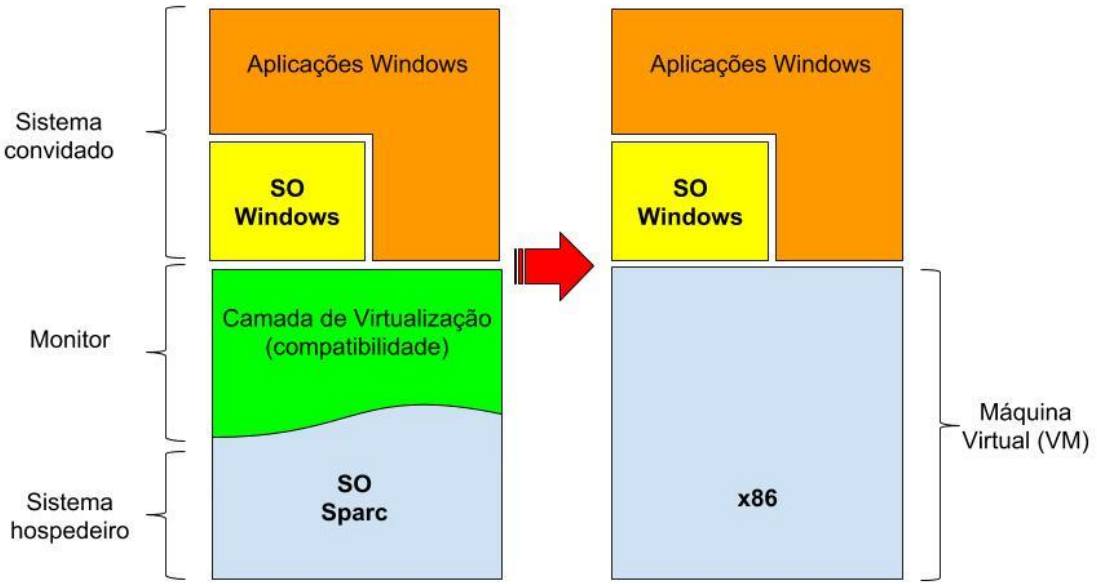


Figura 8 - Visão geral de uma máquina virtual.
Fonte: elaborada pelo autor.

Observando a Figura 8, podemos ver que um ambiente de máquina virtual (VM) consiste de três partes básicas:

a) **o sistema hospedeiro (*host system*)**, ou seja, o sistema real que contém os recursos reais de *hardware* e *software* do sistema;
b) **o sistema convidado (*guest system*)**, ou seja, o sistema virtual que executa sobre o sistema real; em alguns casos, vários sistemas virtuais podem coexistir, executando sobre o mesmo sistema real;
c) **o hipervisor** ou **monitor de virtualização (VMM - *Virtual Machine Monitor*)**, ou seja, a camada de virtualização que constrói as interfaces virtuais a partir da interface real.

Como podemos fazer virtualização em nosso *desktop*?

Separamos um vídeo aula sobre virtualização, preparada pelo Professor Ramos em seu canal. Vamos conhecer conceitos básicos sobre Virtualização e a Instalação do Virtual BOX e do Pacote de Extensão do mesmo.

Vamos aprender um pouco mais sobre o tema?

Acesse:
https://www.youtube.com/watch?v=_7vCg7gKTTU

Essa é apenas uma pitada de virtualização. Veremos mais detalhes nas próximas unidades. Agora, vamos retornar o nosso olhar para as gerências dos sistemas operacionais e aprofundar um pouco mais.

Os sistemas operacionais modernos possuem outras gerências, além dessas que vamos apresentar a seguir. Pode existir gerência de proteção, por exemplo, a qual define políticas de acesso para os sistemas em rede e multiusuários, criando usuários e grupos e suas permissões, além de registrar os recursos por usuários. Podemos ter ainda gerência de energia (dispositivos móveis, por exemplo), gerência de rede e de recursos multimídia. Separamos as mais importantes e comuns a todos os sistemas operacionais para estudar.

2.6 Gerência de processos

A gerência de processos é conhecida também como gerência de processador ou de atividades, pois é responsável por coordenar os processos e atividades em execução. A gerência do processador é necessária para distribuir a capacidade de processamento de forma justa. É importante lembrar que atuar de forma justa é diferente de igual.

A sincronização de atividades, assim como a priorização, é importante para evitar conflitos e dependem de uma comunicação eficiente entre os processos que estão sendo executados.

Sobre a gerência de atividades de um sistema operacional, Maziero (2017, p.27) afirma que:

> "Um sistema de computação **quase sempre tem mais atividades a executar que o número de processadores disponíveis**. Assim, **é necessário criar métodos para multiplexar o(s) processador(es) da máquina entre as atividades presentes**. Além disso, como as diferentes tarefas têm necessidades distintas de processamento, e nem sempre a capacidade de processamento existente é suficiente para atender a todos, **estratégias precisam ser definidas para que cada tarefa receba uma quantidade de processamento que atenda suas necessidades**." (grifo nosso).

Precisamos compreender o conceito de programa e tarefa, pois são conceitos diferentes já que possuem estados diferentes. Vejamos:

De um lado está o programa que representa um conceito estático, e pode ser entendido como um conjunto de uma ou mais sequências de instruções (comandos de linguagem) escritas para resolver um problema específico, constituindo assim um *software*, uma aplicação ou utilitário.

Do outro lado, está a tarefa que representa um conceito dinâmico, pois possui um estado interno definido a cada instante, interagindo com usuários, periféricos ou outras tarefas. A tarefa, executada pelo processador, é compreendida como as sequências de instruções definidas em um programa para realizar seu objetivo.

Sobre a execução de tarefas, Maziero (2017, p. 29) afirma que:

> "[...] em um computador, o processador tem de executar todas as tarefas submetidas pelos usuários. Essas tarefas geralmente têm comportamento, duração e importância distintas. Cabe ao sistema operacional organizar as tarefas para executá-las e decidir em que ordem fazê-lo."

A organização básica do sistema de gerência de tarefas evoluiu conforme evoluíram os sistemas operacionais. Desta forma, temos o sistema monotarefa, o sistema multitarefa e o sistema de tempo compartilhado.

No sistema monotarefa, bastante primitivo, cada programa binário era carregado do disco para a memória e executado até sua conclusão. Sua aplicação era basicamente cálculos numéricos, normalmente com fins militares, e esses sistemas eram dependentes da interação de um operador humano. Para diminuir essa interação surgiu a figura de um programa monitor que era executado para gerenciar a fila de execução de programas, armazenados em disco. O monitor é considerado o precursor dos sistemas operacionais.

Com a evolução do *hardware* surgiram os sistemas multitarefas, já que a velocidade do processador era maior que a velocidade de comunicação com os dispositivos de entrada e saída e o processador ficava ocioso nos momentos de transferência de informação entre a memória e o disco. Assim, a solução foi permitir ao processador suspender uma tarefa enquanto aguardava a transferência de dados externas e passar a executar outra tarefa. Quando finalizasse a transferência de dados da tarefa suspensa o processador retomava a sua execução do ponto que parou. Esse processo de alternância entre tarefas demandava mais memória, e também a necessidade de definir mecanismos para suspender e retomar tarefas. Aqui, surgiu a preempção que é o ato de retirar recursos de uma tarefa (neste caso o processador). Sistemas que implementam esse conceito são conhecidos como *sistemas preemptivos*, e são bem mais produtivos, pois conseguem executar várias tarefas ao mesmo tempo.

Os sistemas de tempo compartilhado surgiram para evitar os erros de *loop infinito* (onde o sistema nunca termina de executar uma tarefa) e para permitir a criação de programas com interatividade. Esse novo conceito de sistema utilizava o *time-sharing* (tempo compartilhado) em que a atenção do processador era dividida entre as tarefas (atividades) por um *quantum* (delta ou taxa) de tempo. Esse conceito permitia que ao término desse tempo uma tarefa perdesse a atenção do processador, e este se dedicasse a olhar a fila de tarefas prontas para execução. Esse conceito toma por base as interrupções geradas pelo temporizador programável do *hardware*.

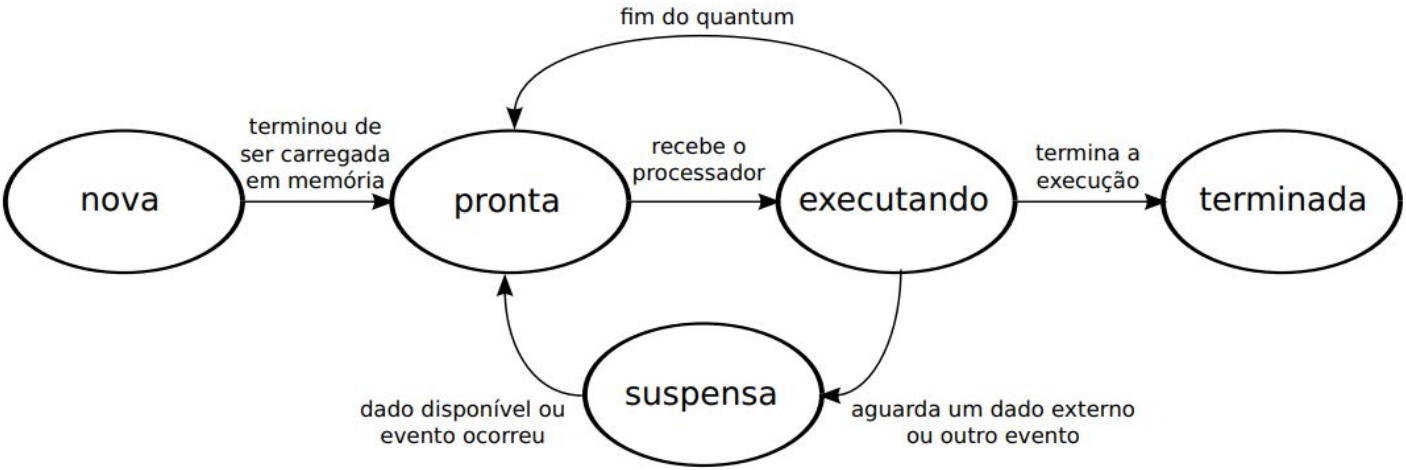


Figura 9 - Diagrama de estados de uma tarefa em um sistema de tempo compartilhado.
Fonte: Maziero (2017, p.33)

Como essas tarefas e processos podem ser vistas e gerenciadas em um sistema operacional real?

No *Linux*, um processo é uma instância de um *software* em execução, ou seja, um programa que está sendo usado. Em outros sistemas, os processos também ganham o nome de tarefas (*tasks*). Quem usou ou usa o *Windows* provavelmente já usou o Gerenciador de Tarefas, um aplicativo que permite, entre coisas, encerrar *softwares* que apresentam comportamentos indesejáveis, fazendo o sistema travar como um todo, por exemplo.

Vamos saber mais?
Acesse:
https://canaltech.com.br/linux/conheca-6-comandos-para-gerenciar-processos-do-linux/

2.7 Gerência de memória

A gerência de memória busca fornecer a cada aplicação em execução um espaço próprio de memória, que seja independente e isolado das demais aplicações. Para viabilizar esse processo de forma eficiente a gerência de memória também faz uso do disco como memória complementar, já que a memória principal normalmente tem tamanho menor. Normalmente, apenas as aplicações prioritárias são mantidas na memória principal, de acesso mais rápido, enquanto as que não estão sendo utilizadas no momento (não tem prioridade) podem ser deslocadas para a memória complementar (discos). É importante ressaltar que esse processo é feito de maneira transparente pelo sistema operacional e a aplicação desconhece o tipo de memória em uso.

Sobre a gerência de memória, Maziero (2017, p. 114) afirma que:

> "A memória principal é um componente fundamental em qualquer sistema de computação. Ela constitui o "espaço de trabalho" do sistema, no qual são mantidos os processos, threads, bibliotecas compartilhadas e canais de comunicação, além do próprio núcleo do sistema operacional, com seu código e suas estruturas de dados. O hardware de memória pode ser bastante complexo, envolvendo diversas estruturas, como caches, unidade de gerência, etc, o que exige um esforço de gerência significativo por parte do sistema operacional. Uma gerência adequada da memória é essencial para o bom desempenho de um computador."

2.8 Gerência de entrada e saída

A gerência de entrada e saída também é conhecida como gerência de dispositivos. Seu papel é gerenciar os diversos dispositivos que podem ser conectados ao sistema operacional como *pen drive*, *disquetes*, discos IDE, SCSI, ATA, SATA e etc… Esse acesso facilitado é realizado por meio de ***drivers*** (ou módulos de kernel no caso do GNU/*Linux*) permitindo o uso de forma comum e transparente.

Sobre a gerência de entrada e saída, Maziero (2017, p. 205) destaca que:

> "Um computador é constituído basicamente de um ou mais processadores, memória RAM e dispositivos de entrada e saída, também chamados de periféricos. **Os dispositivos de entrada/saída permitem a interação do computador com o mundo exterior de várias formas**[...]
>
> Já em ambientes industriais, **é comum encontrar dispositivos de entrada/saída específicos para a monitoração e controle de máquinas e processos de produção**, como tornos de comando numérico, braços robotizados e processos químicos. Por sua vez, o computador embarcado **em um carro conta com dispositivos de entrada para coletar dados do combustível e do funcionamento do motor e dispositivos de saída para controlar a injeção eletrônica e a tração dos pneus,** por exemplo. É bastante óbvio que um computador não tem muita utilidade sem dispositivos periféricos, pois o objetivo básico da imensa maioria dos computadores é receber dados, processá-los e devolver resultados aos seus usuários, sejam eles seres humanos, outros computadores ou processos físicos/químicos externos." (grifo nosso).

2.9 Sistemas de arquivos

A gerência de arquivos é construída sobre a gerência de dispositivos criando uma abstração (tornando transparente e padronizado) de arquivos e diretórios. Essa gerência permite ainda que outros dispositivos possam ser utilizados como arquivos (gravar arquivos em uma saída TCP no UNIX, por exemplo).

Um processo deve ter a capacidade de ler e gravar grande volume de dados em dispositivos de armazenamento de forma permanente. Deve ser capaz, também, de compartilhar esses dados armazenados com outros processos.

Sobre a gerência de arquivos, Maziero (2017, p. 163) afirma que:

> "Um sistema operacional tem por finalidade permitir que os usuários do computador façam a execução de aplicações, como editores de texto, jogos, reprodutores de áudio e vídeo, etc. Essas **aplicações processam informações como textos, músicas e filmes, armazenados sob a forma de arquivos em um disco rígido ou outro meio**." (grifo nosso).

Armazenar e recuperar dados são atividades essenciais para qualquer tipo de aplicação. O sistema operacional (SO) para organizar essas informações de forma estruturada emprega a implementação de arquivos, os quais são gerenciados pelo SO de maneira a facilitar o acesso dos usuários a esses conteúdos.

O sistema de arquivos é a parte mais visível de um SO e a manipulação de arquivos deve ocorrer de forma transparente e uniforme, independente do dispositivo de armazenamento que o SO está utilizando.

A definição de arquivo pode ser compreendida como um conjunto de dados armazenados em um dispositivo físico não volátil, com um nome ou outra referência que permita sua localização posterior. Considerando que um dispositivo de armazenamento pode conter milhões de arquivos é necessário organizar os arquivos em uma estrutura de diretórios que permita o acesso organizado.

Temos um sistema de arquivos quando em um dispositivo fazemos a organização física e lógica dos arquivos e diretórios, ou seja, temos uma imensa estrutura de dados armazenada de forma persistente em um dispositivo físico.

Podemos ter uma diversidade de sistemas de arquivos diferentes. Cada sistema operacional pode utilizar um ou mais sistemas de arquivos diferentes. Como, exemplo, temos o NTFS (nos sistemas *Windows*), Ext2/Ext3/Ext4 (*Linux*), HPFS (MacOS), FFS (Solaris) e FAT (usado em *pendrives* USB, máquinas fotográficas digitais e leitores MP3).

Os atributos de arquivos são informações de controle e dependendo do sistema de arquivos podem variar. Alguns atributos como tamanho, criador, data de criação e proteção, estão presentes na maioria dos sistemas de arquivos. Alguns atributos podem ser modificados apenas pelo sistema operacional, enquanto outros, podem ser modificados pelo usuário.

A depender da forma como os arquivos estão organizados o sistema de arquivos poderá recuperar os arquivos de maneira diferente. O acesso sequencial era utilizado em fitas magnéticas. O acesso direto passou a ser utilizado em discos magnéticos, porém apenas quando os arquivos possuem tamanhos fixos. Finalmente, o acesso indexado, um método mais sofisticado, utiliza índices e chaves para organizar os arquivos. Quando a aplicação precisa de um arquivo ela indica a chave e o SO busca na área de índices, e localiza o ponteiro correto para o arquivo por meio da chave fornecida.

As operações comuns que um sistema de arquivos realiza são: criação, escrita no arquivo, leitura, busca no arquivo, exclusão, truncar um arquivo, anexar e renomear.

A estrutura de diretórios é a forma como o SO organiza logicamente os diferentes arquivos armazenados em um dispositivo físico. Chamamos de diretório a estrutura de dados em que cada entrada guarda informações sobre a localização física, nome, organização e demais atributos. Ao abrir um arquivo o SO procura a sua entrada na estrutura de diretórios, armazenando as informações a respeito dos atributos e localização do arquivo em uma tabela mantida na memória principal.

Um sistema de diretórios normalmente é representado como uma estrutura de diretórios em árvore, conforme a Figura 10.

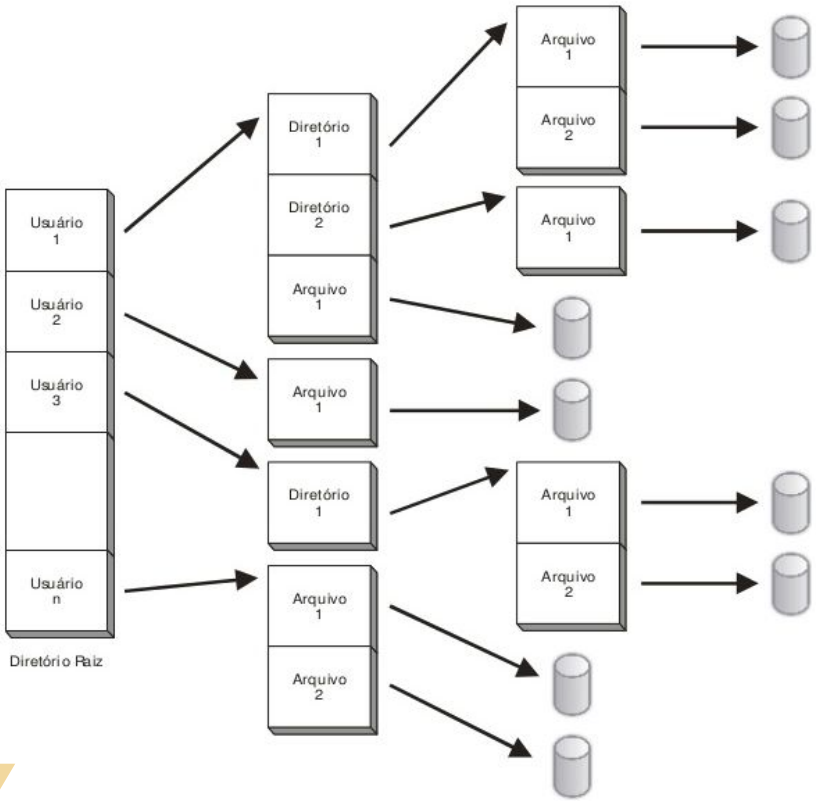


Figura 10 - Estrutura de diretórios em árvore.
Fonte: elaborado pelo autor.

Bora rever!

Essa unidade permitiu aprofundar os conhecimentos sobre os Sistemas Operacionais para computadores e dispositivos móveis. Compreendemos a importância da organização das diversas gerências presentes nos sistemas operacionais. Foi possível observar que todo sistema operacional possui aspectos básicos em sua organização, sendo constituído normalmente por: um núcleo, um gerenciador de memória, um gerenciador de E/S (entradas e saídas), um sistema de arquivos e um processador de comandos/interface com o usuário. Percebemos que os sistemas operacionais podem ser classificados quanto ao número de usuários e, também, quanto ao número de tarefas que podem executar.

Dentro dos sistemas computacionais, cada *hardware* tem suas peculiaridades e cabe ao sistema operacional gerenciar essas diferenças de forma transparente. Por exemplo, um processador de textos (*software*) não necessita saber como ocorre o processo de acesso e gravação de um arquivo (*hardware*). Ele não deve se preocupar em como os dispositivos são acessados.

Compete ao sistema operacional prover interfaces de acesso aos dispositivos, facilitando aos *softwares* formas mais simples de usar esses recursos que as interfaces de baixo nível. Neste processo o sistema operacional tornará os aplicativos independentes do *hardware*, ou seja, ele irá definir interfaces de acesso homogêneas (padronizadas) para dispositivos com tecnologias distintas (diferentes).

Bora rever!

O sistema operacional também é o responsável pela definição das políticas para o gerenciamento do uso dos recursos de *hardware* pelos aplicativos (*softwares*), efetuando a resolução de possíveis disputas e conflitos que possam ocorrer.

Além das funcionalidades básicas oferecidas pela maioria dos sistemas operacionais, várias outras vêm se agregar aos sistemas modernos, para cobrir aspectos complementares, como a interface gráfica, suporte de rede, fluxos multimídia, gerência de energia, etc.

As funcionalidades do sistema operacional geralmente são interdependentes: por exemplo, a gerência do processador depende de aspectos da gerência de memória, assim como a gerência de memória depende da gerência de dispositivos e da gerência de proteção.

Na próxima unidade veremos conceitos básicos do sistema operacional *Android* e como desenvolver aplicativos para diferentes sistemas operacionais. Teremos a oportunidade de aprofundar sobre a arquitetura dos sistemas operacionais para dispositivos móveis (*Android*, *Apple iOS* e *Windows Phone*).

Glossário

***desktop*:** é uma palavra da língua inglesa que designa o ambiente principal do computador. Literalmente, o termo tem o significado de "em cima da mesa". Era frequentemente utilizado para designar um computador de mesa por oposição ao laptop que é o computador portátil.

E/S: entradas e saídas.

***flags*:** é um mecanismo lógico que funciona como semáforo: uma entidade detém como ativa uma determinada *flag* se a característica associada a essa *flag* estiver presente.

interface: é o nome dado para o modo como ocorre a "comunicação" entre duas partes distintas e que não podem se conectar diretamente.

periféricos: periféricos são aparelhos ou placas de expansão que enviam ou recebem informações do computador.

randômico: aleatório, contingente, fortuito. A palavra aleatoriedade exprime quebra de ordem, propósito, causa, ou imprevisibilidade em uma terminologia não científica. Um processo aleatório é o processo repetitivo cujo resultado não descreve um padrão determinístico, mas segue uma distribuição de probabilidade.

registradores: o registrador ou registo de uma CPU é a memória dentro da própria CPU que armazena n bits. Os registradores estão no topo da hierarquia de memória, sendo assim, é uma mídia mais rápida e financeiramente mais custosa de se armazenar dados.

***wireless*:** ou rede sem fio é uma infraestrutura das comunicações sem fio que permite a transmissão de dados e informações sem a necessidade do uso de cabos.

Unidade 3

ARQUITETURA DOS SISTEMAS OPERACIONAIS DE DISPOSITIVOS MÓVEIS

O sistema operacional para dispositivos móveis é um tipo de sistema operacional desenvolvido para *smartphones*, *tablets*, PDAs ou outros. São dispositivos que possuem tela *touch* e uma série de funcionalidades que realmente os tornam móveis e portáteis.

Ainda que existam dispositivos portáteis que possuem algumas características móveis como *laptops* e notebooks, os sistemas operacionais utilizados nesses equipamentos não são considerados móveis, pois são na maioria das vezes, apenas versões comuns de sistemas operacionais já existentes.

Atualmente, devido à convergência tecnológica, estamos passando por um processo tão grande de miniaturização de componentes que essa distinção está se tornando pouco precisa e, alguns sistemas operacionais mais recentes, estão sendo aplicados tanto em computadores quanto em dispositivos móveis, ou seja, feitos para ambos os tipos de equipamentos. É uma forma de tornar única a experiência do usuário que pode utilizar o mesmo sistema operacional em ambos os equipamentos.

Sistemas operacionais móveis, normalmente, combinam recursos úteis para uso móvel ou portátil (tela sensível ao toque, celular, *Bluetooth*, Wi-Fi, GPS de navegação móvel, câmera fotográfica, câmera de vídeo, reconhecimento de voz e leitor de música), com as características dos sistemas operacionais de um computador pessoal.

O funcionamento das aplicações é influenciado pela arquitetura do sistema operacional a qual, também, interfere no processo de desenvolvimento dessas aplicações.

3.1 Arquitetura do Sistema Android

O sistema Android foi desenvolvido com o conceito de plataforma aberta, baseada no sistema operacional GNU/Linux, o que confere aos fabricantes de dispositivos a liberdade de alterar seu código-fonte e, dessa forma, criar versões que melhor se adaptem a seus equipamentos.

O sistema Linux possui diversos componentes, com diversas interfaces gráficas e bibliotecas, assim como disponibiliza muitas ferramentas para a criação de aplicativos.

Além do Google, há um grupo de empresas que contribui com o desenvolvimento do Android, entre elas a Samsung, a LG, a Sony, a Motorola e a HTC.

O mercado de dispositivos Android cresceu muito e Deitel (2015, p. 3) afirma que:

> "A primeira geração de telefones Android foi lançada em outubro de 2008. Em outubro de 2013, um relatório da *Strategy Analytics* mostrou que o Android tinha 81,3% da fatia de mercado global de *smartphones*, comparados com 13,4% da Apple, 4,1% da Microsoft e 1% do Blackberry. De acordo com um relatório do IDC, no final do primeiro trimestre de 2013, o Android tinha 56,5% de participação no mercado global de *tablets*, comparados com 39,6% do iPad da Apple e 3,7% dos *tablets* Microsoft Windows."

O Android é desenvolvido em quatro camadas, cada uma contendo funcionalidades distintas, sendo a zero a de nível mais baixo, e a três a de nível mais alto.

Observe a seguir as descrições das camadas e componentes da plataforma Android que podem ser visualizadas na Figura 1.

- **Camada 0 (Kernel)** – É onde estão as funções de gestão básica da máquina, ou seja, controle da memória, gerenciamento dos componentes de *hardware* e segurança. O Kernel do GNU/Linux é a base da plataforma Android. O Android *Runtime* (ART) utiliza o kernel para realizar as funções de gerenciamento e encadeamento de memória de baixo nível;

- **Camada 1 (*Hardware Abstraction Layer* - HAL)** - Essa camada provê interfaces padronizadas que disponibilizam as capacidades de hardware do dispositivo para a estrutura da Java API de maior nível. 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*;

- **Camada 2 (Bibliotecas do sistema operacional)** – É responsável pelas funcionalidades básicas do dispositivo. As bibliotecas incluem um conjunto de programas em linguagem Java que podem ser desenvolvidos pelos desenvolvedores de aplicativos. Vários componentes e serviços principais do sistema Android, como ART e HAL, são implementados por código nativo que exige bibliotecas nativas programadas em C e C++. A plataforma Android fornece a Java *Framework* APIs para expor a funcionalidade de algumas dessas bibliotecas nativas aos aplicativos.

Por exemplo, é possível acessar *OpenGL* ES pela Java *OpenGL* API da estrutura do Android para adicionar a capacidade de desenhar e manipular gráficos 2D e 3D no seu aplicativo. Para dispositivos com Android versão 5.0 (API nível 21) ou mais recente, cada aplicativo executa o próprio processo com uma instância própria do Android *Runtime* (ART).

- **Camada 3 (Java API Framework)** – É totalmente acessível aos programadores e nela estão as funcionalidades de gestão das informações dos aplicativos, como notificações e demais recursos. O conjunto completo de recursos do SO Android está disponível pelas APIs programadas na linguagem Java. Essas APIs formam os blocos de programação que você precisa para criar os aplicativos Android simplificando a reutilização de componentes e serviços de sistema modulares e principais, inclusive: um sistema de visualização rico e extensivo útil para programar a interface de usuário (IU) de um aplicativo, com listas, grades, caixas de texto, botões e, até mesmo, um navegador da web incorporado; um gerenciador de recursos, fornecendo acesso a recursos sem código como strings localizados, gráficos e arquivos de *layout*; um gerenciador de notificação que permite que todos os aplicativos exibam alertas personalizados na barra de *status*; um gerenciador de atividade que gerencia o ciclo de vida dos aplicativos e fornece uma pilha de navegação inversa; provedores de conteúdo que permitem que aplicativos acessem dados de outros aplicativos, como o aplicativo Contatos, ou compartilhem os próprios dados. Os desenvolvedores têm acesso completo aos mesmos *Frameworks* APIs que os aplicativos do sistema Android usam;

- **Camada 4 (Aplicativos do sistema)** - É a camada que se comunica com o usuário. Nela estão os aplicativos como os de mensagens, e-mails, mapas e navegadores. O Android vem com um conjunto de aplicativos principais para e-mail, envio de SMS, calendários, navegador de internet, contatos etc. Os aplicativos inclusos na plataforma não têm status especial entre os aplicativos que o usuário opta por instalar. Portanto, um aplicativo terceirizado pode se tornar o navegador da Web, o aplicativo de envio de SMS ou até mesmo o teclado padrão do usuário (existem algumas exceções, como o aplicativo Configurações do sistema). Os aplicativos do sistema funcionam como aplicativos para os usuários e fornecem capacidades principais que os desenvolvedores podem acessar pelos próprios aplicativos. Por exemplo, se o seu aplicativo quiser enviar uma mensagem SMS, não é necessário programar essa funcionalidade — é possível invocar o aplicativo de SMS que já está instalado para enviar uma mensagem ao destinatário que você especificar.

Para desenvolver aplicativos para a plataforma Android, é necessária a utilização do Android SDK (*Software Development Kit*). O Android SDK constitui um conjunto de ferramentas necessárias ao desenvolvedor, incluindo um emulador, que permite testar as aplicações em um PC, antes de enviá-las ao dispositivo móvel.

Além do emulador, o Android SDK inclui as bibliotecas requeridas para o desenvolvimento, uma ferramenta de depuração de erros, exemplos de códigos, documentação e tutoriais para o sistema operacional Android.

Toda vez que uma nova versão do sistema Android é liberada, o correspondente kit Android SDK também é disponibilizado.

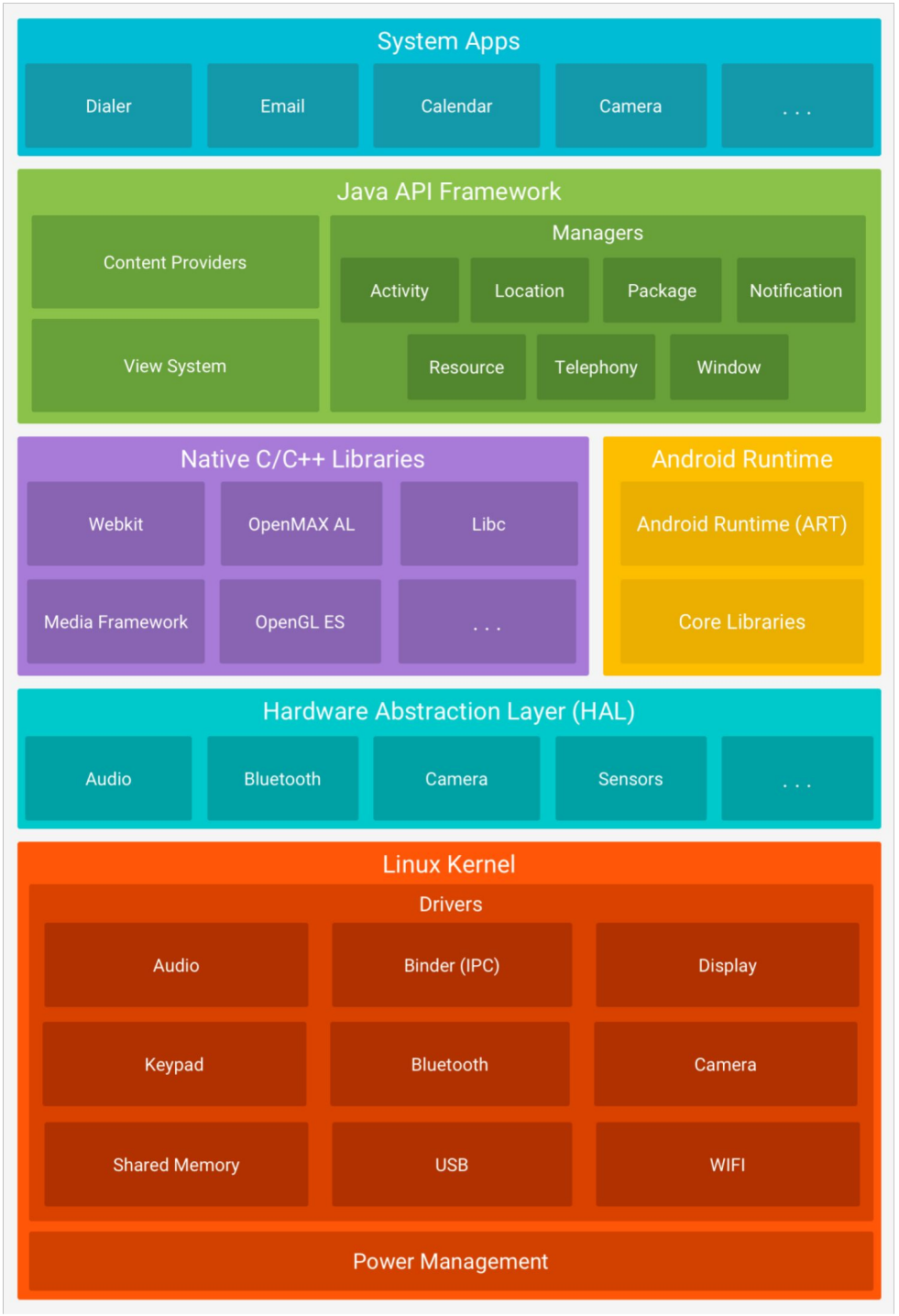


Figura 1 - Camadas e componentes da plataforma Android.
Fonte: Adaptado de *Android Developers* - Disponível em:
https://developer.android.com/guide/platform/images/android-stack_2x.png?hl=pt-br

A responsabilidade por gerenciar os processos e *threads*, além da memória, arquivos, pastas, redes, drivers dos dispositivos e energia é da camada mais baixa da arquitetura Android, o kernel Linux.

A arquitetura do sistema Android executa todos os componentes de uma aplicação em um mesmo processo e *thread*. Quando outro processo faz requisição de memória, ou, quando a memória fica sobrecarregada, por ter maior nível de importância, o processo em execução é parado e o processo que tem maior importância de acordo com o usuário é carregado. Dentro da arquitetura Android há cinco níveis de importância de processos:

1. *Foreground Process*;
2. Processo visível;
3. Processo de serviço;
4. Serviços em *Background*;
5. Processos vazios.

Para realizar a gerência de processos o Android utiliza o ***binder***, um mecanismo que faz a comunicação entre os processos. Qualquer comunicação entre os processos passa sempre pelo *binder* (uma classe do Java). O escalonamento de CPU é feito com a criação de filas com os componentes: processos interativos, processos em *batch*, processos em tempo real. Dentro da arquitetura Android o escalonador emprega o conceito de *time-sharing*, e é do tipo preemptivo.

A arquitetura do sistema Android não implementa nenhuma proteção para travamento de ***Deadlock***, partindo do princípio que eles não irão ocorrer.

***Deadlock*:** no contexto de sistemas operacionais, refere-se a uma situação em que ocorre um impasse, e dois ou mais processos ficam impedidos de continuar suas execuções - ou seja, ficam bloqueados, esperando uns pelos outros.

O gerenciamento de memória do Android toma por base o gerenciamento de memória do GNU/Linux. Todas as operações em níveis mais baixos, como I/O e gerência de memória são tratadas pelo LINUX.

Assim, da mesma forma que o Linux, para o gerenciamento de memória é utilizada a técnica de memória virtual por dois motivos básicos:

- e forma segura e eficiente em permitir o compartilhamento da memória entre diversos programas;
- remover de uma quantidade pequena e limitada da memória principal os transtornos de programação;

Assim, como a maioria dos sistemas operacionais por meio da gerência de arquivos facilita o acesso dos usuários ao seu conteúdo. O sistema de arquivos é responsável, dentro do sistema operacional, por realizar essa gerência.

De forma resumida os principais diretórios do Android são:

- ***data*** - armazena os dados das aplicações;
- ***system*** - armazena informações do sistema;
- ***system/lib*** - armazena as bibliotecas;
- ***system/bin e system/xbin*** - armazena os serviços;
- ***system/app*** - armazena as aplicações java.

A biblioteca denominada Bionic foi implementada pelo Android para ser utilizada como biblioteca do sistema. Essa biblioteca contém os seguintes diretórios:

- **/** – diretório raiz (Android e Linux);
- **/Cache** – armazenamento de dados para execuções rápidas (Android);
- **/Data** – Dados (Android). Esse contém dados do usuário armazenado sem uma partição separada de MTD;
- **/Default.prop** - (Android) definições de propriedade-padrão e valores restaurados a partir dos arquivos em cada reinicialização;
- **/Dev** – arquivos de dispositivos (Linux e Android).

A maioria dos diretórios do sistema operacional Android segue o padrão do sistema operacional GNU/Linux. Apenas alguns poucos diretórios foram adicionados. Vejamos:

- **/etc** – arquivos de configuração (Linux e Android);
- **/init** – inicialização (Android);
- **/lib** - Bibliotecas essenciais partilhadas (Linux e Android);
- **/lost+found** - arquivos recuperados (Linux e Android);
- **/media** - mídias removíveis (Linux e Android);
- **/proc** – kernel e arquivos de processo (Linux e Android);
- **/root** - Diretório home para o super usuário (Linux e Android);
- **/sbin** – arquivos binários de administração (Linux e Android);
- **/sdcard** – Cartão SD (Android);
- **/system** – Sistema (Android);
- **/tmp** - arquivos temporários (Linux e Android).

Como posso saber mais a respeito do desenvolvimento Android?

Na página oficial do Android, é possível encontrar as ferramentas de desenvolvimento atualizadas, incluindo o Android Studio, que reúne as ferramentas do SDK em um ambiente de trabalho para o desenvolvedor. Se estiver desenvolvendo um aplicativo que exige código C ou C++, você pode usar o Android NDK para acessar algumas dessas bibliotecas de plataforma nativa diretamente do seu código nativo.

Para obter mais informações, acesse a página oficial do Android: https://developer.Android.com

3.2 Arquitetura do Sistema Apple iOS

O sistema iOS tem seu desenvolvimento sob responsabilidade da Apple e é bastante protegido, isto é, raramente algum aplicativo se comunica diretamente com o *hardware* do dispositivo. Os aplicativos devem ser desenvolvidos para se comunicar por meio de interfaces de sistema. Essa é uma das razões pelas quais o sistema iOS é considerado um dos mais seguros, e também, imune a ataques e vírus. Assim como o Android, o iOS está organizado em quatro camadas, sendo a Core OS a de nível mais baixo, e a *Cocoa Touch* a de nível mais alto.

Camada *Core OS* – É a camada de comunicação direta com a máquina, com acesso aos componentes de *hardware*, aos acessórios conectados e à execução de cálculos fundamentais.

Camada *Core Services* – Nesta camada, estão os serviços básicos do sistema para uso das aplicações. No entanto, a maioria dessas aplicações não usa esses serviços diretamente, mas sim partes do sistema construídas "em cima" desses serviços.

Camada Média – Nesta camada estão os recursos e as tecnologias de áudio, vídeo e gráficos, todos projetados para tratar aplicativos multimídia e tornar sua programação mais fácil.

Camada *Cocoa Touch* – Nesta camada encontram-se os recursos para a elaboração das aplicações. Esse nível dispõe de tecnologias como a de notificação e a de multitarefa.

O desenvolvedor de aplicativos para iOS necessita trabalhar em um ambiente de desenvolvimento próprio, o XCode IDE (*Integrated Development Environment*), parte do iOS SDK (*Software Development Kit*).

Para realizar o desenvolvimento, são utilizadas linguagens próprias como o *Swift*, além do *Objective-C*.

Opcionalmente, o desenvolvedor pode utilizar outras plataformas, como a versão iOS do Xamarin, que também oferece os recursos necessários ao desenvolvimento e aos testes de aplicativos.

Como desenvolver para iOS?

O desenvolvimento para iOS segue ainda requisitos rigorosos da loja de aplicativos, que determinam funcionalidades que podem ou não existir, bem como diretrizes de desenho da interface.

Para obter mais informações, acesse a página oficial de desenvolvimento da Apple. Acesse:
https://developer.apple.com/

3.3 Arquitetura do Windows Phone

Ao contrário do que muitos pensam o *Windows Phone* não é uma continuação do *Windows Mobile*. A Microsoft, em completa ruptura com as antigas versões, busca agora focar diretamente no mercado consumidor, apresentando uma nova interface gráfica.

O objetivo da Microsoft é manter, em todos os dispositivos, a estrutura de menu chamada Metro, que foi lançada na versão 8 do Windows e está presente no Windows 10 com pequenas modificações.

Assim como o Android e o iOS, a estrutura interna do Sistema operacional Windows é baseada em camadas, cada uma com funções específicas.

As camadas do Windows são quatro, sendo a de mais baixo nível conhecida como camada *Hardware*, e a de mais alto nível como plataforma de aplicações.

Camada *Hardware* – Esta é a camada que se comunica com o dispositivo diretamente, seja com componentes internos da máquina, seja com acessórios externos.

Camada HAL – Com nível mais alto que a camada *Hardware*, o objetivo da camada HAL (*Hardware Abstraction Layer*) é possibilitar o acesso indireto dos aplicativos aos componentes da máquina. Nesta camada, estão as funcionalidades como a gestão da inicialização, além de recursos básicos de entrada e saída de informações.

Camada Kernel – Esta camada contém funcionalidades de gestão de memória, rede, segurança e os controladores dos dispositivos de *hardware*.

Camada *Application Plataforms* – Esta camada é onde estão as interfaces com o usuário e nela é executada a maior parte das funcionalidades das aplicações.

O desenvolvimento para a plataforma Windows é facilitado pela utilização do *Visual Studio Community* e do SDK (*Software Development Kit*) do Windows, que apoiam a elaboração dos chamados aplicativos universais para a plataforma.

Em tese, em um mesmo ambiente, é possível desenvolver aplicativos para computadores, *tablets* e *smartphones* equipados com Windows.

Como desenvolver para o Windows Phone?

É importante ter algum conhecimento em C# ou VB, que são as duas possíveis linguagens de desenvolvimento, para iniciar o desenvolvimento para Windows Phone.

Para obter mais informações, acesse a página oficial de desenvolvimento da Microsoft. Acesse:
https://developer.microsoft.com

3.4 Desenvolvimento de Aplicativos para Diferentes Sistemas Operacionais

Os sistemas iOS, Android e Windows têm características diferenciadas que, somadas aos diversos dispositivos existentes, produzem uma enormidade de variações com que os desenvolvedores têm de lidar. Os programadores que conseguem merecem um brinde!

Essa enorme diversidade de dispositivos móveis no mercado impõe grandes desafios aos programadores, que têm de desenvolver e testar seus aplicativos em diferentes equipamentos, com:

- telas de tamanhos variados;
- processadores de capacidades diversas;
- dispositivos de *hardware* específicos;
- quantidades imprevisíveis de memória disponível;

O desenvolvimento de aplicações para dispositivos *mobile* tem características diferentes do desenvolvimento tradicional, devido à variedade de equipamentos disponíveis e às limitações da capacidade de processamento, do tamanho da tela e da área de trabalho.

Além disso, esses dispositivos estão sempre sujeitos a configurações diferenciadas tanto de *hardware* quanto de *software* por parte dos fabricantes, principalmente quando estes fazem modificações no sistema operacional, o que é muito comum nos dispositivos Android.

A capacidade de processamento depende, fundamentalmente, de detalhes construtivos do dispositivo, como modelo do processador, velocidade das vias de comunicação entre os componentes de *hardware*, memória disponível para cálculos e outras especificações que estão totalmente fora do controle do desenvolvedor.

Com tantos desafios a serem vencidos, será que é possível atender todas as plataformas?

Essa é uma questão muito importante para o desenvolvedor, que deve entender e aceitar o seguinte: muito dificilmente, conseguiremos atender todas as plataformas.

Dessa forma, o desenvolvedor terá de escolher para quais sistemas operacionais irá desenvolver. Lembrando que, não basta desenvolver um aplicativo, é necessário cuidar de seu suporte e de suas atualizações, e isso pode ser muito trabalhoso se houver a intenção de atingir todos os dispositivos do mercado.

Mesmo grandes empresas, que têm centenas de desenvolvedores trabalhando para elas, fazem escolhas. São raros os aplicativos disponíveis para todas as plataformas, mesmo considerando as três dominantes: Android, iOS e Windows.

Os recursos de desenvolvimento utilizados nas aplicações móveis devem assegurar certo nível de suporte a essa diversidade. Dessa forma, devem garantir a execução sem erros e o controle de comportamentos indesejáveis do aplicativo em diferentes aparelhos.

Um dos focos do desenvolvedor deve ser a usabilidade, que é a facilidade, ou simplicidade, com que ocorre a interação de um aplicativo ou *website* com o usuário.

No contexto do desenvolvimento de aplicações móveis, a usabilidade envolve desde a apresentação gráfica do aplicativo, suas respostas aos comandos do usuário, seu nível de interatividade e em que grau sua utilização pode ser considerada intuitiva.

Telas simples e com poucas opções facilitam o entendimento e a utilização dos aplicativos.

Já vimos algumas das características dos três sistemas operacionais mais utilizados no mundo dos dispositivos móveis, certo?

Falamos também, um pouco a respeito dos ambientes de desenvolvimento. Mas, agora, vamos detalhar um pouco mais as funcionalidades desses ambientes, também chamados de plataformas.

Para facilitar a vida do desenvolvedor, os próprios fornecedores de sistemas operacionais disponibilizam ambientes integrados de desenvolvimento ou IDE (*Integrated Development Environment*).

Onde obter informações sobre os ambientes de desenvolvimento para dispositivos móveis?

Os ambientes oficiais do Android, iOS e Windows podem ser obtidos, gratuitamente, nos seguintes sites:

https://developer.Android.com

https://developer.apple.com/

https://developer.microsoft.com/

Funcionalidades dos Ambientes de Desenvolvimento

Os ambientes reúnem ferramentas para apoiar o desenvolvedor em seu trabalho.

Vejamos a seguir as ferramentas típicas em desenvolvimento:

a) Editor

Programa para se trabalhar os códigos fonte escritos nas linguagens de programação aceitas pelo sistema. É possível editar os códigos fonte em editores padrões, como o editor de textos do Windows. No entanto, os editores integrados aos ambientes de desenvolvimento, usualmente, dispõem de mais recursos para organização do programa fonte e sua documentação, facilitando o trabalho do desenvolvedor;

b) Compilador

Transforma o código fonte do programa escrito pelo desenvolvedor em um programa em linguagem de máquina, isto é, em um formato que será entendido pelo dispositivo;

c) Gerador de Testes

Possibilita realizar testes automáticos de entrada e saída de dados no aplicativo, simulando a operação realizada pelos usuários. É um importante recurso quando se deseja testar diversas possibilidades de inserção de dados, buscando eventuais erros ou grandes volumes de acesso;

d) Depurador (ou debugger, em inglês)

Ajuda a encontrar erros e a aprimorar o programa escrito pelo desenvolvedor;

e) Emulador

Ferramenta que permite simular a execução do aplicativo em dispositivos móveis dentro do ambiente de desenvolvimento, normalmente em um PC ou Mac. Por meio dos emuladores, é possível ver e testar o aplicativo como se ele estivesse em um dispositivo móvel, sendo possível realizar ajustes e correções sem a necessidade de publicá-lo para carga nos dispositivos;

f) Ferramentas para distribuição

Tem por objetivo formatar o "pacote" de informações, documentos e executáveis, que será direcionado à loja do fornecedor (*AppStore* da Apple, Google Play ou Windows *Store*) para a aprovação e publicação do aplicativo.

Além dos ambientes de desenvolvimento oficiais, existem outros no mercado, que podem ser utilizados pelos desenvolvedores de acordo com a disponibilidade e facilidade de uso.

Na verdade, a escolha do ambiente de desenvolvimento é, muitas vezes, uma questão de gosto pessoal ou de familiaridade do desenvolvedor, exceto quando o desenvolvedor atua em uma empresa ou um projeto em que determinado ambiente é mandatório, para fins de padronização e documentação.

O desenvolvedor tem disponíveis diversos ambientes de desenvolvimento para cada plataforma.

Para o Android, além do IDE oficial Android *Studio*, podem ser usados os seguintes ambientes:

- Eclipse (https://eclipse.org/ide/)
- NetBeans (http://plugins.netbeans.org/plugin)
- Xamarin (https://www.xamarin.com/)

A linguagem de programação para desenvolvimento de aplicativos para Android é o Java, mas alguns códigos em C e C++ podem ser utilizados.

A plataforma Android dispõe de muitos recursos. Dessa forma, desenvolvedores experientes podem utilizar mais de um ambiente, aproveitando os recursos que consideram melhores de um ou de outro.

O Apple iOS dispõe do ambiente oficial *Xcode*, mas o desenvolvedor também pode optar por utilizar um ambiente de desenvolvimento alternativo, como um dos seguintes:

- Jetbrains (https://www.jetbrains.com/objc/)
- Xamarin (https://www.xamarin.com/studio)

A linguagem de programação para desenvolvimento de aplicativos para o iOS é o *Swift*, uma linguagem de programação criada pela Apple para as plataformas Mac e iOS.

A interface gráfica dos aplicativos é criada pelo *Xcode*. No entanto, o desenvolvedor pode utilizar outras linguagens, como o *Objective-C*, por exemplo.

O Windows dispõe do ambiente oficial *Visual Studio*, mas também podem ser usados os seguintes ambientes, por exemplo:

- Eclipse (https://eclipse.org/ide/)
- Xamarin (https://www.xamarin.com/studio

As linguagens de programação para o desenvolvimento de aplicativos para Windows são o C, C#, C++ e *Visual Basic*.

A estratégia de desenvolvimento da Microsoft conta com a utilização de um único conjunto de ferramentas para os programadores de aplicativos para PC, *tablet* e *smartphone*.

Bora rever!

Essa unidade apresentou a arquitetura dos principais Sistemas Operacionais para computadores e dispositivos móveis. Foi possível compreender que os sistemas operacionais móveis normalmente combinam recursos úteis para uso móvel ou portátil (tela sensível ao toque, celular, Bluetooth, Wi-Fi, GPS de navegação móvel, câmera fotográfica, câmera de vídeo, reconhecimento de voz e leitor de música) com as características dos sistemas operacionais de um computador pessoal.

O funcionamento das aplicações é influenciado pela arquitetura do sistema operacional a qual, também, interfere no processo de desenvolvimento dessas aplicações.

O sistema Android foi desenvolvido com o conceito de plataforma aberta, baseada no sistema operacional GNU/Linux, o que confere aos fabricantes de dispositivos a liberdade de alterar seu código-fonte e, dessa forma, criar versões que melhor se adaptem a seus equipamentos. O sistema Linux possui diversos componentes, com diversas interfaces gráficas e bibliotecas, assim como disponibiliza muitas ferramentas para a criação de aplicativos.

O sistema iOS tem seu desenvolvimento sob responsabilidade da Apple e é bastante protegido, isto é, raramente algum aplicativo se comunica diretamente com o hardware do dispositivo. Os aplicativos devem ser desenvolvidos para se comunicar por meio de interfaces de sistema. Essa é uma das razões pelas quais o sistema iOS é considerado um dos mais seguros, e, também, imune a ataques e vírus.

Bora rever!

Assim como o Android e o iOS, a estrutura interna do Sistema operacional Windows é baseada em camadas, cada uma com funções específicas. As camadas do Windows são quatro, sendo a de mais baixo nível conhecida como camada Hardware, e a de mais alto nível como plataforma de aplicações.

Os sistemas iOS, Android e Windows têm características diferenciadas que, somadas aos diversos dispositivos existentes, produzem uma enormidade de variações com que os desenvolvedores têm de lidar.

O desenvolvimento de aplicações para dispositivos mobile tem características diferentes do desenvolvimento tradicional, devido à variedade de equipamentos disponíveis e às limitações da capacidade de processamento, do tamanho da tela e da área de trabalho.

Na próxima unidade, veremos conceitos de virtualização, os tipos e como construir máquinas virtuais, além de aprofundar um pouco mais nos conceitos da máquina virtual Android.

Glossário

Touch: é uma tela sensível ao toque é um tipo de ecrã sensível à pressão, dispensando, assim, a necessidade de outro periférico de entrada de dados, como o teclado;

***Runtime*:** é o período em que um programa de computador permanece em execução;

API: Interface de Programação de Aplicação (português brasileiro) é um conjunto de rotinas e padrões estabelecidos por um *software* para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do *software*, mas apenas usar seus serviços;

***Framework*:** é uma abstração que une códigos comuns entre vários projetos de *software* provendo uma funcionalidade genérica;

***OpenGL*:** o *OpenGL* é uma API livre utilizada na computação gráfica, para desenvolvimento de aplicativos gráficos, ambientes 3D, jogos, entre outros;

***Layout*:** é uma palavra inglesa, muitas vezes usada na forma portuguesa "leiaute", que significa plano, arranjo, esquema, design, projeto;

***Software Development Kit*:** é um conjunto de ferramentas de desenvolvimento de *software* que permite a criação de aplicativo para certo pacote de *software*, *hardware* ou sistema operacional;

***Ahead-of-time*:** adiantado, antecipado;

***Just-in-time*:** significa “na hora certa” ou "momento certo";

***Bytecodes*:** um formato de código intermediário entre o código fonte, o texto que o programador consegue manipular, e o código de máquina, que o computador consegue executar.

Unidade 4

VIRTUALIZAÇÃO

Virtualizar significa: simular, mascarar ambientes isolados, capazes de rodar diferentes sistemas operacionais (SO) dentro de uma mesma máquina. Este processo aproveita ao máximo a capacidade do *hardware*, que muitas vezes fica ociosa em determinados períodos do dia, da semana ou do mês.

A possibilidade de fornecer ambientes de execução independentes a diferentes usuários em um mesmo equipamento físico, concomitantemente, permite um aproveitamento maior. A virtualização ajuda, também, a resolver o problema da restrição ao uso do *hardware* para a utilização por determinados *softwares*, pois ela diminui o poder dos sistemas operacionais. Alguns *softwares* só rodam sobre o sistema operacional para o qual foram desenvolvidos.

Assim, a virtualização permite que diferentes sistemas rodem em um mesmo equipamento (máquina), ampliando, desta forma, a quantidade de *softwares* que podem ser empregados em um mesmo *hardware*.

Em servidores, essa abordagem é bastante empregada, com a vantagem adicional de fornecer uma interface (camada de abstração) dos verdadeiros recursos de uma máquina, já que provê um *hardware* virtual para cada sistema. No ambiente de servidores ela é uma excelente ferramenta para migração de sistemas.

Vamos refletir um pouco sobre o surgimento da virtualização. Considerando que o *hardware*, o sistema operacional e as aplicações foram desenvolvidos ao longo do tempo por empresas diferentes, e em momentos históricos diferentes.

É natural que com o passar do tempo os caminhos do desenvolvimento de tecnologias computacionais acabassem por se tornar heterogêneos (distintos) e por consequência, surgissem sistemas incompatíveis. Esse processo levou ao surgimento de plataformas operacionais diferenciadas, porém, incompatíveis entre si. Com o advento da virtualização surgiu a possibilidade da criação de uma máquina virtual (VM), uma camada de compatibilidade sobre o sistema operacional hospedeiro e hardware real, que poderia simular o ambiente de outra máquina real, onde, então, seria possível instalar outro sistema operacional convidado.

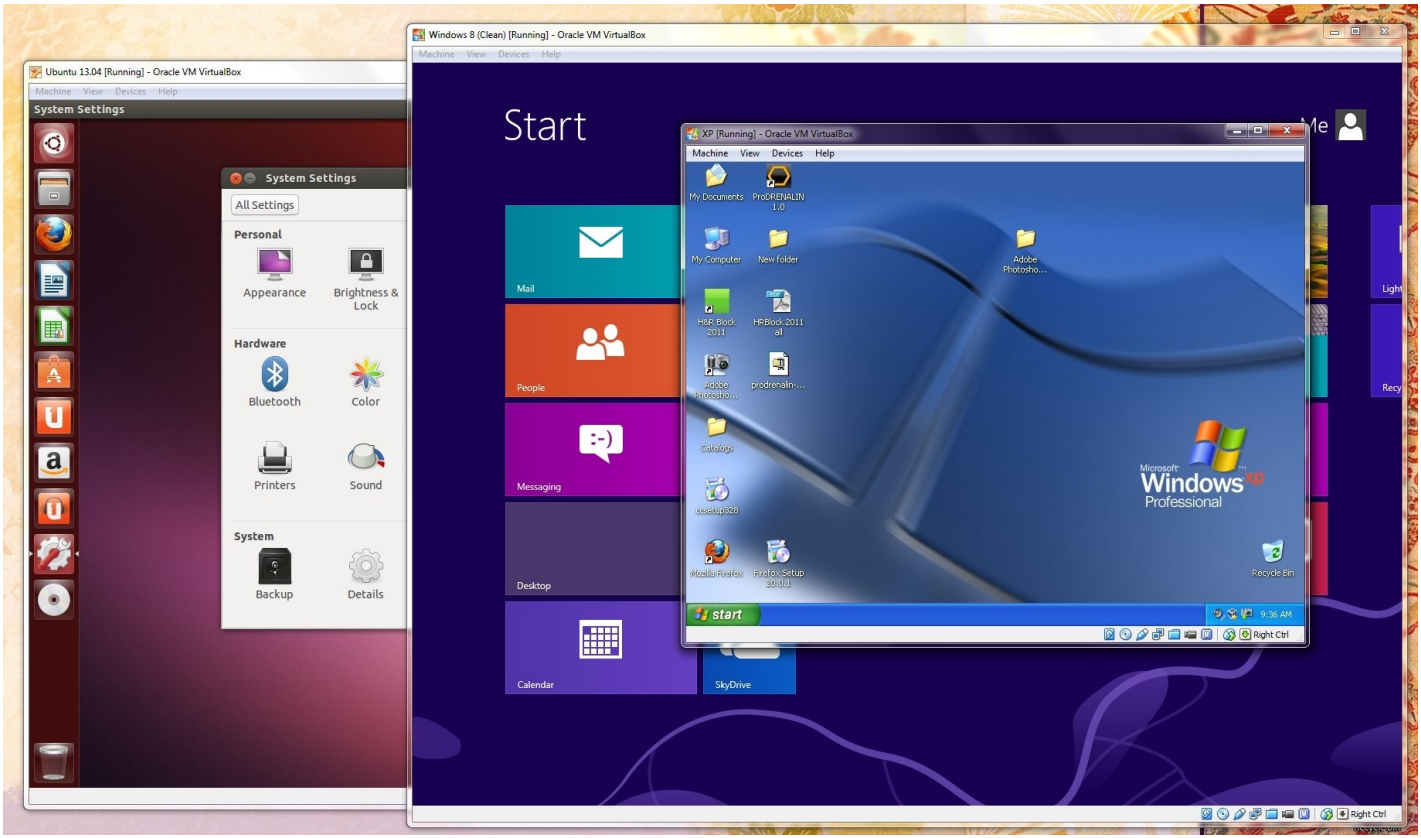


Figura 1 - Máquinas virtuais - Virtualbox.
Fonte: https://www.servti.com/blog/wp-content/uploads/2016/09/virtualbox.jpg

A Figura 1 exemplifica a execução de 3 (três) máquinas virtuais sobre um *hardware* x86 com Windows. O *Virtualbox* está executando três máquinas virtuais com *hardwares* independentes do *hardware* real, criados pela camada de virtualização, o que permite executar diversos sistemas operacionais diferentes.

Desta forma, com a virtualização, uma técnica que possibilita a criação de diversos ambientes virtuais (máquinas virtuais), pode instalar e executar diversos sistemas operacionais (SO) em uma mesma máquina real, permitindo executar um SO diferente dentro de cada máquina virtual criada. Para Deitel (2010, p.15), "uma máquina virtual (VM) é uma abstração em *software* de um computador executado frequentemente como uma aplicação de usuário sobre o sistema operacional nativo".

4.1 Fundamentos da virtualização

Uma máquina real (física) é composta por diversos dispositivos físicos que disponibilizam operações para as camadas de abstração acima. O núcleo desse sistema é o processador e o *chipset* da placa-mãe traz consigo todos os recursos como áudio, vídeo, memória, e portas de comunicação. Cada *hardware* pode ter as suas peculiaridades, mas de forma geral é assim que o *hardware* está organizado.

O emulador é um *software* que tem a função de enganar uma aplicação. A Figura 2 demonstra a atuação do emular. Imagine uma máquina real que executa certa aplicação (X) em um determinado sistema operacional (A). Se desejarmos executar essa aplicação em outra máquina, a qual não possui os mesmos recursos, com outro *hardware* e sistema operacional diferente (B). Desejamos obter o mesmo resultado da execução da aplicação X no sistema operacional A.

A Figura 2 demonstra a técnica para rodar uma aplicação projetada para um sistema operacional em outro incompatível.

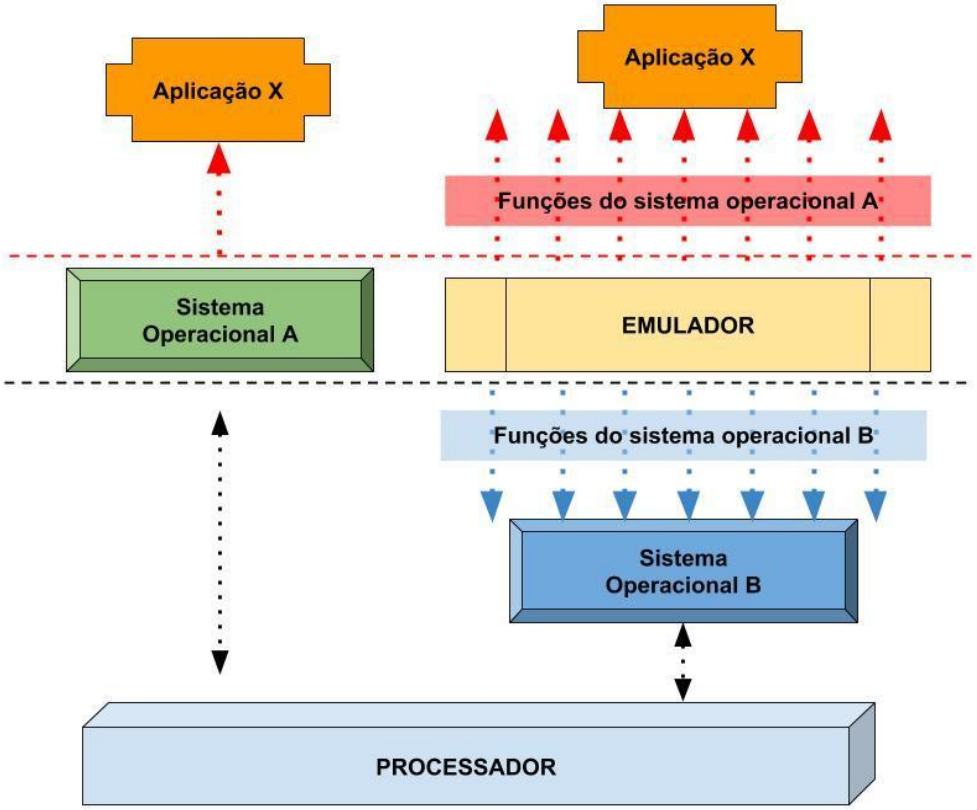


Figura 2 - Sistema de emulação.
Fonte: elaborado pelo autor.

Utilizando *software*, ou seja, uma máquina virtual (VM), podemos criar a imitação de uma máquina real. A simulação (imitação) criada por ela permite rodar sistemas operacionais que terão a ilusão de estar rodando na máquina real que a máquina virtual está simulando. Esse tipo de imitação pode ser realizada (criadas) em nível de sistema operacional, ou, até mesmo de aplicação. Vejamos:

a) **máquina virtual criada a nível de aplicação:** a máquina virtual é executada sobre o sistema operacional, denominado anfitrião. O nível acima da VM irá acreditar que abaixo dele há uma máquina real (imitação criada pela VM), e, assim, pode-se rodar outro sistema operacional diferente do que está abaixo da imitação. Esse é o processo empregado para executar uma aplicação que foi projetada para o sistema A em um sistema B (incompatível);

b) **máquina virtual criada a nível de sistema operacional:** neste caso, emprega-se um Monitor de Máquina Virtual (MMV), em inglês *Virtual Machine Monitor* (VMM). A função do MMV é controlar o *hardware* e permitir a criação de várias máquinas virtuais. O MMV permite assim executar vários sistemas operacionais, um em cada imitação criada. Cada sistema operacional novo, sendo executado, imaginará estar rodando sobre um *hardware* real, quando na verdade, está sobre uma interface de abstração criada pela VM para simular o *hardware* real, compatível com esse novo SO. A vantagem desse processo é permitir executar diferentes SO sobre um mesmo *hardware* (mesma máquina real), cada um com suas aplicações, de forma transparente e independente. O MMV, também, faz a gerência do uso de dispositivos físicos de entrada e saída, recursos de memória da máquina real, permitindo que os sistemas operacionais que estão sendo executados nas VMs acessem esses recursos de forma transparente e sem conflitos. É claro que o *hardware* real não será duplicado, mas sim multiplexado (compartilhado).

É desta maneira que os recursos da máquina são compartilhados entre os usuários. Eles permanecerão acreditando que estão executando suas aplicações diretamente na máquina real, quando na verdade estão compartilhando (dividindo) os recursos disponíveis.

Na figura a seguir temos a comparação de um sistema não virtualizado com os dois tipos possíveis de virtualizações.

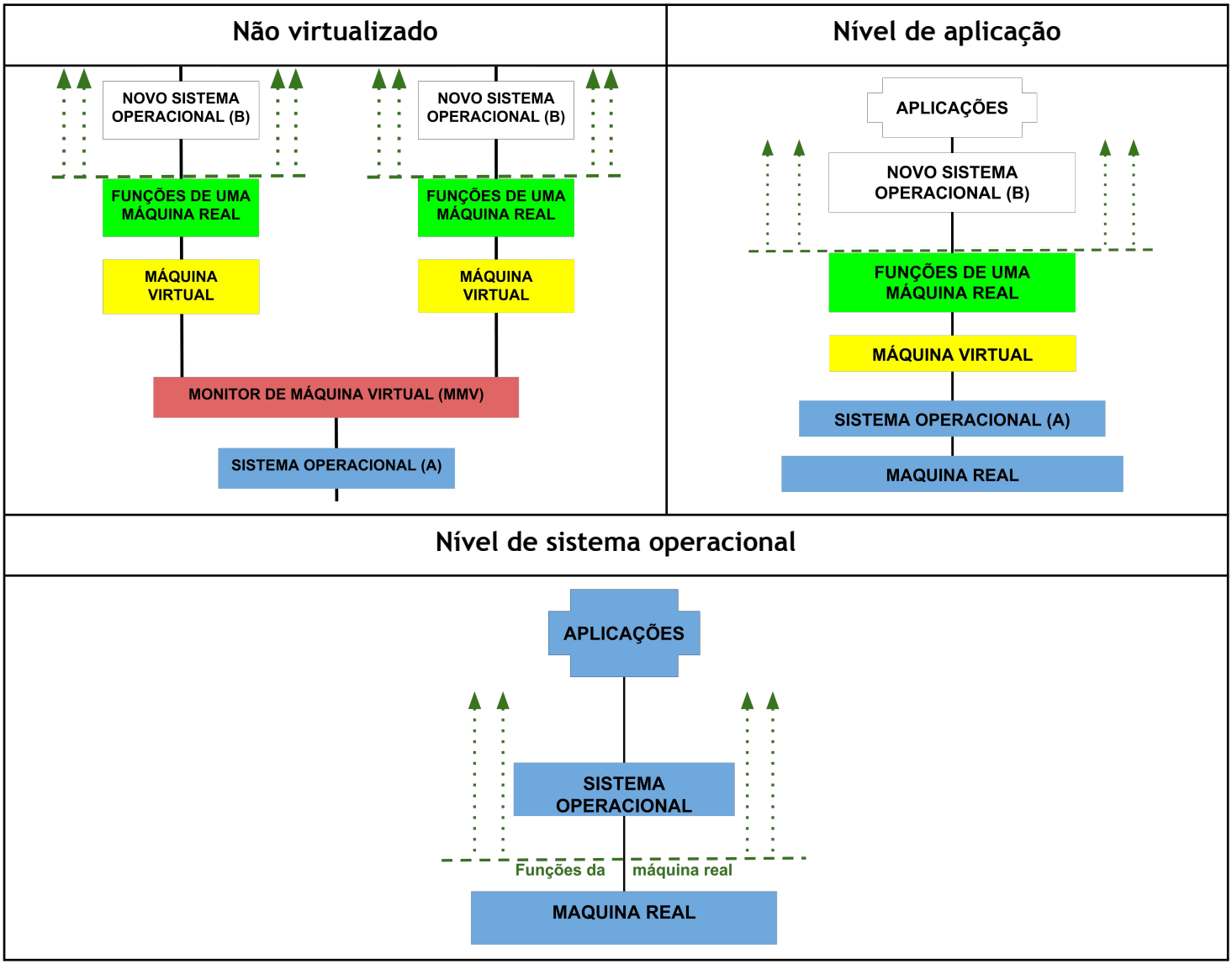


Figura 3 - Exemplo de sistema não virtualizado com possíveis virtualizações.
Fonte: elaborado pelo autor.

4.2 A construção de máquinas virtuais

A virtualização consiste na possibilidade de gerar abstrações de recursos (reais ou virtuais) empregando *softwares*, de forma que eles pareçam ser diferentes do que realmente são. Esse conceito pode ser estendido e empregado na execução de virtualização de três formas diferentes: virtualização de sistema operacional, virtualização de *hardware* e virtualização de linguagem de programação.

Vejamos:

a) **virtualização de sistema operacional:** é implantada sobre um sistema operacional existente e consiste em criar a simulação de um novo sistema operacional. Mesmo não havendo grandes ganhos significativos, permite resolver a necessidade de executar aplicações em um sistema operacional incompatível. Temos como exemplo, o *FreeBSD Jail* e o *User-mode Linux*;

b) **virtualização de *hardware*:** é utilizada para imitar um *hardware* real. A VM é executada sobre um SO e permite a execução de outros SO sobre ela. O sistema que está logo abaixo da VM pode ser um MMV ou um SO direto na máquina real. O *VMware* e o *Xen* na plataforma x86 são exemplos desse tipo de virtualização;

c) **virtualização de linguagens de programação:** é empregada para fazer o computador real se comportar de forma diferente, ou seja, com novas instruções. A VM irá executar o programa de acordo com o comportamento definido pelo usuário para a virtualização, e ficará encarregada de traduzir as ações em novas ações que o sistema operacional abaixo compreenda. Como exemplo desse tipo de virtualização, temos o Java e o *Smalltalk*.

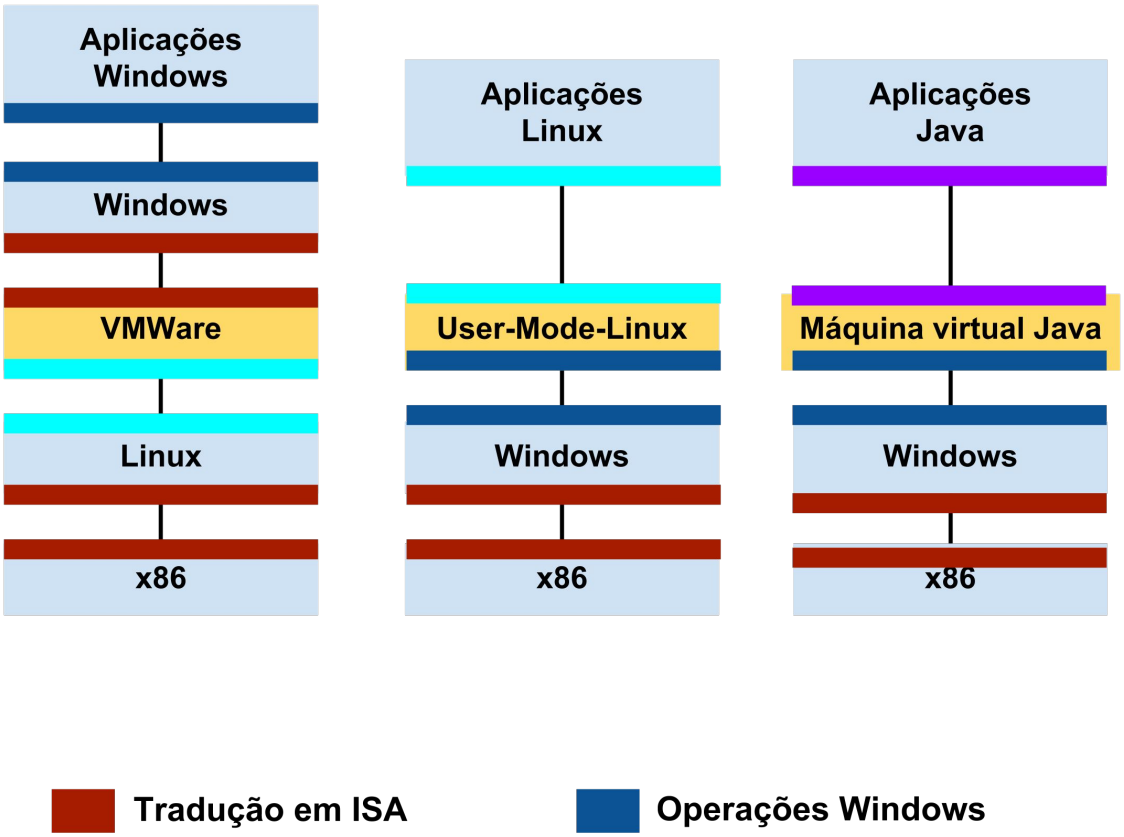


Figura 4 - Sistema de emulação.
Fonte: UFRJ - GTA - Virtualização. Disponível em: https://www.gta.ufrj.br/grad/09_1/versao-final/virtualizacao/figuras/como_e_empregado.jpg

A virtualização pode ser empregada para facilitar o desenvolvimento de *softwares* e de sistemas operacionais. Podemos utilizar VMs para testar o SO em desenvolvimento, sem causar danos ou impactos. De igual forma, os *softwares* em desenvolvimento podem ser testados em sistemas virtuais.

A grande vantagem do emprego de máquinas virtuais, nesse contexto de desenvolvimento, é que podemos simular ambientes e testar sistemas operacionais e *softwares* que ainda estão em desenvolvimento, antes de serem comercializados. Outro aspecto é permitir a comparação de execução de um determinado *software* ou aplicação, em diversos sistemas operacionais, empregando a mesma máquina real.

É possível, ainda, executar aplicações diversas na mesma máquina, pois não é raro que alguém necessite rodar aplicações voltadas para sistemas operacionais diferentes. Neste caso, a virtualização pode ser empregada para permitir rodar vários sistemas diferentes na mesma máquina, e, assim, várias tarefas projetadas para sistemas incompatíveis podem ser executadas no mesmo ambiente de *hardware*.

Outra aplicação a considerar é empregar VMs para criar situações e ambientes simulados (criar a ilusão de recursos reais) que permitam avaliar o funcionamento de aplicações ou sistemas operacionais. Neste caso, a VM é empregada para simular, testar e avaliar resultados de forma simulada, atendendo a situações reais.

É possível empregar virtualização para manter sistemas legados (com funções muito específicas e de difícil adaptação) que necessitam versões antigas ou específicas de aplicações ou sistemas operacionais.

A virtualização pode ser empregada, também, para a consolidação de servidores, uma vez que a rápida evolução do poder computacional não foi acompanhada pela demanda por capacidade. Esse processo permitiu recursos ociosos, como servidores utilizando uma pequena parte do seu poder de processamento. Desta forma, a virtualização atua para permitir que um servidor empregue toda a sua capacidade hospedando diversos sistemas operacionais e aplicações ao mesmo tempo. Então, usar o máximo da capacidade da máquina, significa na prática, diminuir custos com *hardware*.

Máquinas virtuais (virtualização) possibilitam o fornecimento de serviços dedicados a clientes específicos, traduzindo-se em processos mais seguros com confiabilidade e disponibilidade.

Na área de segurança podemos empregar máquinas virtuais para atuar como *honeypots* contra ataques de invasores na Internet ou Intranet. É possível manter de propósito várias instâncias de VMs para que os invasores possam atacá-las. Esse tipo de máquina de "isca" é chamado de *honeypots* (pote de mel) e tem a função de servir como sistema de monitoramento dos possíveis ataques, ajudando a desenvolver meios de prevenção contra eles.

Cinco motivos pelos quais você deve usar o recurso da virtualização de sistemas

Virtualizar sistemas operacionais é um recurso bastante utilizado por Administradores de Redes em *DataCenters*. Um motivo óbvio para tal uso é, sem dúvidas, ter a possibilidade de gerenciar diversos sistemas em apenas um computador hospedeiro. Mas existem cinco motivos pelos quais você também deveria usar esse recurso em sua máquina doméstica! Descubra por que!

Vamos ler um pouco mais sobre o tema?

Visite:
https://www.linuxdescomplicado.com.br/2011/06/5-motivos-pelos-quais-voce-deve-usar-o-2.html

4.3 Tipos de máquinas virtuais

A virtualização pode ser classificada quanto à arquitetura do processo. Temos três tipos possíveis. Vejamos:

a) **Arquitetura Tipo I** - Neste formato de arquitetura o Monitor de Máquina Virtual (MMV ou VMM) é implementado diretamente sobre o *hardware* hospedeiro. Assim, o monitor pode controlar todas as operações de acesso requisitadas pelos sistemas convidados, simulando máquinas físicas com propriedades distintas e trabalhando de forma isolada. Neste formato podemos ter diferentes sistemas operacionais virtuais operando sobre o mesmo *hardware*. São exemplos desse tipo: VMM's XEN e VMWARE ESX SERVER.

Figura 5 - Arquitetura Tipo I.
Fonte: Elaborado pelo autor.

b) Arquitetura Tipo II - caracteriza-se pela implementação do Monitor de Máquina Virtual (MMV ou VMM) sobre o sistema operacional instalado no *hardware* anfitrião, operando como um processo desse sistema operacional. As operações que seriam controladas pelo sistema operacional do hospedeiro são simuladas pelo monitor para as máquinas virtuais. Temos como exemplos desse tipo: *VMWARE SERVER* e *VIRTUALBOX*.

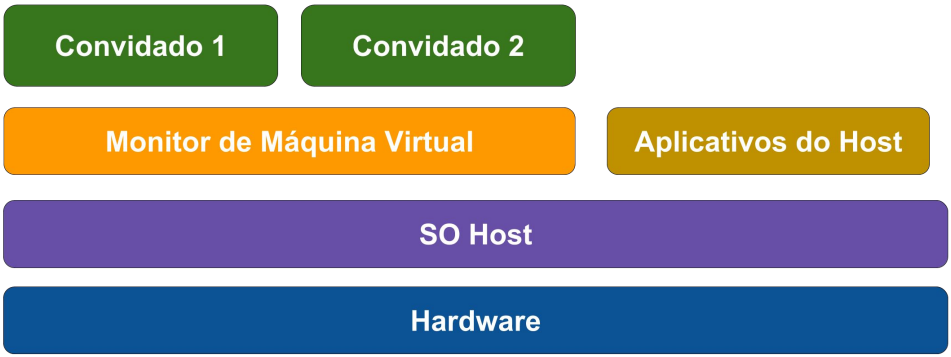


Figura 6 - Arquitetura Tipo II.
Fonte: Elaborado pelo autor.

c) Arquitetura Híbrida - este tipo engloba qualidades dos dois tipos anteriores. É possível agregar características do tipo I ao tipo II e vice-versa. O formato híbrido busca otimizar os tipos I e II.

Otimização para MMV de Tipo I: O sistema convidado acessa diretamente o *hardware*, através de modificações no sistema convidado e no monitor. Essa otimização é utilizada para algumas funcionalidades do Xen.

Otimização para MMV de Tipo II: o sistema convidado acessa diretamente o SO real da máquina, sobre o qual funciona o monitor. Dessa forma, alguns sistemas virtuais não precisam ser inteiramente providos pelo monitor. No *VMware*, o sistema de arquivos do SO real é utilizado pelo sistema convidado, poupando o monitor de gerar um sistema similar na aplicação virtual.

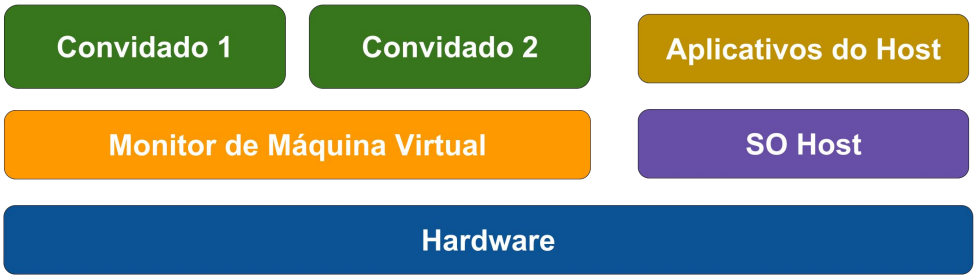


Figura 7 - Arquitetura Híbrida.
Fonte: Elaborado pelo autor.

4.4 Técnicas de virtualização

a) **Virtualização completa (total):** como o nome sugere, uma estrutura completa de *hardware* é virtualizada. O *hardware* hospedeiro é completamente abstraído e todas as características de um equipamento virtual são emuladas, ou seja, todas as instruções solicitadas pelo sistema convidado são interpretadas no Monitor de Máquina Virtual. O Sistema convidado não precisa sofrer qualquer tipo de alteração.

O sistema hospedado ignora a existência da máquina virtual e opera como se funcionasse diretamente sobre o sistema operacional para o qual foi projetado para funcionar. Há um grande nível de compatibilidade, porém há perda de velocidade.

A virtualização completa é mais flexível em termos de SO convidados, uma vez que este não precisa ser modificado para implementação dessa técnica. Todas as instruções são interpretadas pelo monitor de máquina virtual. Em compensação, essa interpretação de cada instrução provoca perda de desempenho de processamento, uma vez que o monitor de máquina virtual se utiliza de dispositivos de virtualização que atendem a uma gama de aplicativos e, por isso, possuem uma baixa especialização. Assim, não é possível ter o máximo desempenho desse aplicativo.

b) Paravirtualização: nessa técnica, a máquina virtual não é idêntica ao equipamento físico original, para que o sistema hospedado possa enviar as instruções mais simples diretamente para o *hardware*, restando apenas as instruções de nível mais alto para serem interpretadas pelo MMV. Há perda de compatibilidade, porém com ganho de velocidade.

Esse procedimento requer que o sistema operacional convidado seja modificado para interagir com o MMV e selecionar quais instruções devem ser interpretadas nele ou diretamente no hardware hospedeiro.

A paravirtualização é menos flexível, pois carece de modificações no sistema operacional convidado, para que este possa trabalhar perfeitamente. Porém, o fato de o sistema operacional convidado saber que opera sobre uma máquina virtual e, com isso, mandarem as instruções mais simples diretamente para o *hardware* diminui a sobrecarga no MMV e permite uma maior especialização dos dispositivos de virtualização. Dessa forma, os aplicativos operam mais próximos de sua capacidade máxima, melhorando seu desempenho em comparação à virtualização completa. Além disso, na paravirtualização, a complexidade das máquinas virtuais a serem desenvolvidas diminui consideravelmente.

c) Recompilação Dinâmica: na recompilação dinâmica, as instruções são traduzidas durante a execução do programa da seguinte forma: as instruções do programa são identificadas em forma de sequência de bits. Em seguida, as sequências são agrupadas em instruções mais próximas do sistema operacional hospedeiro. Por último, essas instruções são reagrupadas em um código de mais alto nível, que, por sua vez, é compilado na linguagem nativa do sistema hospedeiro. A recompilação dinâmica tem como principal vantagem a melhor adequação do código gerado ao ambiente de virtualização, que, com a compilação durante a execução, pode refletir melhor o ambiente original do aplicativo. Isso acontece porque durante a execução, há novas informações disponíveis, às quais um compilador estático não teria acesso. Dessa forma, o código gerado se torna mais eficiente. Em contrapartida, essa técnica exige maior capacidade de processamento, visto que a recompilação acontece em tempo real de execução do programa.

4.5 Ambientes de máquinas virtuais

Conheceremos alguns dos sistemas disponíveis para trabalhar com máquinas virtuais em sistemas operacionais convencionais (*desktops* e servidores). Apresentaremos os sistemas *VMWare*, *Virtualbox*, FreeBSD Jails, Xen, *User-Mode Linux*, QEMU, Valgrind e JVM.

Eles são exemplos de máquinas virtuais de aplicação e de sistema, empregando virtualização total ou paravirtualização, além de abordagens híbridas. A escolha foi feita considerando que eles estão entre os mais representativos de suas respectivas classes.

Sistema	Descrição
VMWare	(Virtualização total) - Atualmente, o *VMware* é a máquina virtual para a plataforma x86 de uso mais difundido, provendo uma implementação completa da interface x86 ao sistema convidado. Embora essa interface seja extremamente genérica para o sistema convidado, acaba conduzindo a um hipervisor mais complexo. Como podem existir vários sistemas operacionais em execução sobre mesmo *hardware*, o hipervisor tem que emular certas instruções para representar corretamente um processador virtual em cada máquina virtual, fazendo uso intensivo dos mecanismos de tradução dinâmica. **Site:** https://www.vmware.com/br.html
Virtualbox	(Virtualização total) - O *VirtualBox* é uma ferramenta de virtualização originalmente desenvolvida pela a empresa alemã Innotek. Posteriormente, foi adquirida pela Sun, que por sua vez em 2010 tornou-se aquisição da *Oracle Corporation*, recebendo assim, uma nova nomenclatura: *Oracle VM VirtualBox*. **Site:** https://www.virtualbox.org/
FreeBSD Jails	(Virtualização no nível do Sistema Operacional) - O sistema operacional *FreeBSD* oferece um mecanismo de confinamento de processos denominado *Jails*, criado para aumentar a segurança de serviços de rede. Esse mecanismo consiste em criar domínios de execução distintos (denominados *jails* ou celas). Cada cela contém um subconjunto de processos e recursos (arquivos, conexões de rede) que pode ser gerenciado de forma autônoma, como se fosse um sistema separado. Cada domínio é criado a partir de um diretório previamente preparado no sistema de arquivos. Um processo que executa a chamada de sistema *jail* cria uma nova cela e é colocado dentro dela, de onde não pode mais sair, nem seus filhos. **Site:** https://www.freebsd.org/doc/handbook/jails.html

Xen	(Paravirtualização) - O ambiente *Xen* é um hipervisor nativo para a plataforma x86 que implementa a paravirtualização. Ele permite executar sistemas operacionais como Linux e Windows especialmente modificados para executar sobre o hipervisor. Versões mais recentes do sistema Xen utilizam o suporte de virtualização disponível nos processadores atuais, o que torna possível a execução de sistemas operacionais convidados sem modificações, embora com um desempenho ligeiramente menor que no caso de sistemas paravirtualizados. **Site:** https://www.xenproject.org
User-Mode Linux	(Virtualização Total) - O *User-Mode Linux* foi proposto como uma alternativa de uso de máquinas virtuais no ambiente *Linux*. O núcleo do *Linux* foi portado de forma a poder executar sobre si mesmo, como um processo do próprio Linux. O resultado é um *user space* separado e isolado na forma de uma máquina virtual, que utiliza dispositivos de *hardware* virtualizados a partir dos serviços providos pelo sistema hospedeiro. Essa máquina virtual é capaz de executar todos os serviços e aplicações disponíveis para o sistema hospedeiro. Além disso, o custo de processamento e de memória das máquinas virtuais *user-Mode Linux* é geralmente menor que aquele imposto por outros *hipervisores* mais complexos. **Site:** http://user-mode-linux.sourceforge.net/
QEMU	(Virtualização completa) - O QEMU é um hipervisor que não requer alterações ou otimizações no sistema hospedeiro, pois utiliza intensivamente a tradução dinâmica como técnica para prover a virtualização. É um dos poucos hipervisores recursivos, ou seja, é possível chamar o QEMU a partir do próprio QEMU. **Site:** https://www.qemu.org/

Valgrind	(Virtualização de Aplicações - compilação *just-in-time* (JIT)) - O *Valgrind* é uma ferramenta de depuração de uso da memória RAM e problemas correlatos. Ele permite investigar vazamentos de memória (*memory leaks*), acessos a endereços inválidos, padrões de uso dos caches e outras operações envolvendo o uso da memória RAM. O *Valgrind* foi desenvolvido para plataforma x86 Linux, mas existem versões experimentais para outras plataformas. **Site:** http://www.valgrind.org/
JVM	(Virtualização de Aplicações) - É comum a implementação do suporte de execução de uma linguagem de programação usando uma máquina virtual. Um bom exemplo dessa abordagem ocorre na linguagem Java. Tendo sido originalmente concebida para o desenvolvimento de pequenos aplicativos e programas de controle de aparelhos eletroeletrônicos, a linguagem Java mostrou-se ideal para ser usada na Internet. O que a torna tão atraente é o fato de programas escritos nessa linguagem de programação poder ser executados em praticamente qualquer plataforma. A virtualização é o fator responsável pela independência dos programas Java do *hardware* e dos sistemas operacionais: um programa escrito em Java, ao ser compilado, gera um código binário específico para uma máquina abstrata denominada máquina virtual Java (JVM - Java Virtual Machine). A linguagem de máquina executada pela máquina virtual Java é denominada *bytecode* Java, e não corresponde a instruções de nenhum processador real. A máquina virtual deve então interpretar todas as operações do *bytecode*, utilizando as instruções da máquina real subjacente para executá-las. **Site:** https://www.oracle.com/technetwork/java/index.html

4.6 Máquina Virtual Android (Dalvik, ART (Android Runtime)).

Quem é o responsável pela execução de aplicativos instalados no sistema operacional do seu computador ou smartphone? A resposta será idêntica e simples se estivermos falando de um computador comum ou celular com iOS ou Windows Phone. Quem executa essa função de executar aplicativos instalados é o SO. Simples.

Porém, quando falamos de Android a questão muda um pouco. Devido a grande variedade de equipamentos e componentes diferentes que cada um possui o sistema operacional do Google (Android) precisa usar uma máquina virtual, para criar uma camada de compatibilidade (uma interface de abstração).

É necessário compreender, então, que o Android utilizou uma **máquina virtual** chamada **Dalvik** até a versão KitKat, e depois foi descontinuada, sendo substituída pela **máquina virtual ART**.

***KitKat*:** As versões do sistema operacional Android foram apresentadas na unidade 3 de nosso material didático. Para saber mais faça a releitura desse tópico.

Dentro das camadas da Arquitetura Android temos a camada de Runtime, onde está à máquina virtual Dalvik, criada para cada aplicação executada no Android. Esse novo modelo de Máquina Virtual possui melhor desempenho, maior integração com a nova geração de *hardware* e foi projetada para executar vários processos paralelamente.

Na máquina virtual Dalvik as aplicações escritas em Java são compiladas em *bytecodes* Dalvik e executadas usando a Máquina Virtual Dalvik (JIT (*Just-In-Time*)), que é uma máquina virtual especializada desenvolvida para uso em dispositivos móveis, o que permite que programas sejam distribuídos em formato binário (*bytecode*) e possam ser executados em qualquer dispositivo Android, independentemente do processador utilizado. Apesar das aplicações Android serem escritas na linguagem Java, ela não é uma máquina virtual Java, já que não executa *bytecode* JVM.

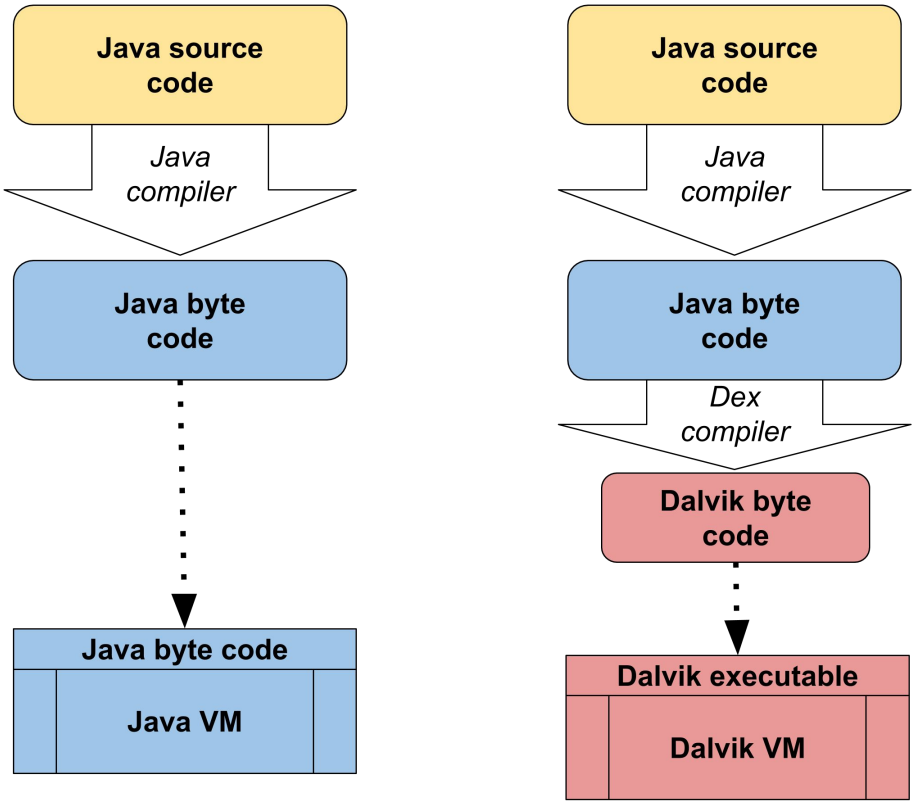


Figura 8 - Comparação entre JVM tradicional e VM Dalvik.
Fonte: Site Luiztools - Acesso em 01 Dez 2018. Disponível em:
http://www.luiztools.com.br/wp-content/uploads/2016/07/android-dalvik-300x223.png

Mas, afinal, o que é o Android *RunTime* (ART) então? O ART (*Ahead-of-Time Compilation*) é projetado para executar várias máquinas virtuais em dispositivos de baixa memória executando arquivos DEX, um formato de *bytecode* projetado especialmente para Android, otimizado para oferecer consumo mínimo de memória. É possível construir cadeias de ferramentas, como Jack, e compile fontes Java em *bytecodes* DEX, que podem ser executadas na plataforma Android. Alguns dos recursos principais de ART são:

- Compilação "*ahead-of-time*" (AOT) e "j*ust-in-time*" (JIT);
- Coleta de lixo (GC) otimizada;
- Melhor compatibilidade de depuração, inclusive um gerador de perfil de exemplo, exceções de diagnóstico detalhadas e geração de relatórios de erros, além da capacidade de definir pontos de controle para monitorar campos específicos.

Antes do Android versão 5.0 (API nível 21), o Dalvik era o tempo de execução do Android. Se o seu aplicativo executa o ART bem, deve funcionar no Dalvik também, mas talvez não vice-versa.

O Android também contém um conjunto das principais bibliotecas de tempo de execução que fornecem a maioria da funcionalidade da linguagem de programação Java, inclusive alguns recursos de linguagem Java 8 que a estrutura da Java API usa.

Palestra - *Campus Startup School*: Desvendando o ecossistema e a arquitetura Android

Nesta palestra, Neto Marin ensina os conceitos básicos de desenvolvimento para Android, incluindo a evolução da plataforma, como otimizá-la em todas as telas e dominar o processo de arquitetura do aplicativo.

Visite:
https://www.youtube.com/watch?v=hiJrgrujWeA

Bora rever!

Essa unidade, apresentou os fundamentos da virtualização. Foi possível perceber a importância de das técnicas de virtualização e como são organizados os ambientes de máquinas virtuais. Vimos, ainda, o funcionamento da máquina virtual Android (Dalvik e ART).

Virtualizar significa: simular, mascarar ambientes isolados, capazes de rodar diferentes sistemas operacionais (SO) dentro de uma mesma máquina. Este processo aproveita ao máximo a capacidade do *hardware*, que muitas vezes fica ociosa em determinados períodos do dia, da semana ou do mês.

A virtualização permite a criação de uma máquina virtual (VM), funcionando como uma camada de compatibilidade sobre o sistema operacional hospedeiro e *hardware* real, que pode simular o ambiente de outra máquina real, onde então é possível instalar outro sistema operacional convidado.

Utilizando uma máquina virtual (VM) podemos criar a imitação de uma máquina real. A simulação (imitação) criada por ela permite rodar sistemas operacionais que terão a ilusão de estar rodando na máquina real que a máquina virtual está simulando. Esse tipo de imitação pode ser realizado (criadas) como virtualização de sistema operacional, virtualização de *hardware* ou virtualização de linguagens de programação.

Bora rever!

A virtualização pode ser empregada para facilitar o desenvolvimento de *softwares* e sistemas operacionais. Podemos utilizar VMs para testar o SO em desenvolvimento, sem causar danos ou impactos. De igual forma os *softwares* em desenvolvimento podem ser testados em sistemas virtuais.

Podemos realizar virtualização empregando alguns sistemas disponíveis para trabalhar com máquinas virtuais em sistemas operacionais convencionais (*desktops* e servidores) como *VMWare*, *Virtualbox*, *FreeBSD Jails*, *Xen*, *User-Mode Linux*, *QEMU*, *Valgrind* e *JVM*. Eles são exemplos de máquinas virtuais de aplicação e de sistema, empregando virtualização total ou paravirtualização, além de abordagens híbridas.

E assim, chegamos ao fim do conteúdo desse capítulo. Revise os conteúdos sempre que tiver dúvidas sobre o tema.

Glossário

Chipset: é um conjunto de componentes eletrônicos de baixa capacidade, em um circuito integrado, que gerencia o fluxo de dados entre o processador, memória e periféricos. É normalmente encontrado na placa mãe.

***Hipervisor*:** um *hipervisor*, ou monitor de máquina virtual, é um *software*, *firmware* ou *hardware* que cria e roda máquinas virtuais (VMs). O computador no qual o *hipervisor* roda uma ou mais VMs é chamado de máquina hospedeira (*host*), e cada VM é chamada de máquina convidada (*guest*).

***Ahead-of-time*:** o ato de transformar o código-fonte em código de máquina é chamado de "compilação". Quando todo o código é transformado de uma só vez antes de atingir as plataformas que o executam, o processo é chamado de Compilação AOT (*Ahead-Of-Time*).

***Just-in-time*:** em Ciência da Computação, compilação *just-in-time* (JIT), também conhecida como tradução dinâmica, é a compilação de um programa em tempo de execução , usando uma abordagem diferente da compilação anterior à execução.

JVM: (do inglês *Java Virtual Machine* - JVM) é um programa que carrega e executa os aplicativos Java, convertendo os *bytecodes* em código executável de máquina. A JVM é responsável pelo gerenciamento dos aplicativos, à medida que são executados.

Interface: uma interface, em ciência da computação, é a fronteira que define a forma de comunicação entre duas entidades.

Abstração: é a habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais.

Referências Bibliográficas

LECHETA, R. R. **Android Essencial**. São Paulo: Novatec, 2016.

LECHETA, R. R. **Google Android**. São Paulo: Novatec, 2016.

DEITEL, P.; DEITEL, H.; DEITEL, A. **Android para programadores**. 2a ed. Porto Alegre: Bookman, 2015.

TANENBAUM, Andrew S. **Sistemas Operacionais Modernos**. 3a. ed. São Paulo: Pearson Pretice Hall, 2010.

MACHADO, Francis B. **Arquitetura de Sistemas Operacionais.** 4a. ed. Rio de Janeiro: LTC, 2011.

SILBERSCHATZ, Abraham. **Fundamentos de Sistemas Operacionais**. Rio de Janeiro: LTC, 2011.

OLSEN, Diogo Roberto. **Sistemas Operacionais**. Curitiba: Editora Livro Técnico, 2010.

INSTITUTO FEDERAL
Brasília

Secretaria de
Educação Profissional
e Tecnológica

Ministério da
Educação