

SISTEMAS E APLICAÇÕES DISTRIBUÍDAS

Prof. Carlos Andre de Sousa Rocha

Prof.º Muriel de Fátima Bernhardt



Indaial – 2020

2ª Edição



Copyright © UNIASSELVI 2020

Elaboração:

Prof. Carlos Andre de Sousa Rocha

Prof.^a Muriel de Fátima Bernhardt

Revisão, Diagramação e Produção:

Centro Universitário Leonardo da Vinci – UNIASSELVI

Ficha catalográfica elaborada na fonte pela Biblioteca Dante Alighieri

UNIASSELVI – Indaial.

R672s

Rocha, Carlos Andre de Sousa

Sistemas e aplicações distribuídas. / Carlos Andre de Sousa Rocha; Muriel de Fátima Bernhardt. – Indaial: UNIASSELVI, 2020.

255 p.; il.

ISBN 978-65-5663-098-4

1. Sistemas distribuídos. - Brasil. 2. Sistemas operacionais. – Brasil. I. Bernhardt, Muriel de Fátima. II. Centro Universitário Leonardo Da Vinci.

CDD 004

APRESENTAÇÃO

Caro acadêmico, seja bem-vindo! A partir de agora você iniciará um novo desafio envolvendo o estudo sobre os Sistemas de Aplicações Distribuídas. Este livro didático foi elaborado de modo a apresentar o que são e como funcionam os sistemas distribuídos e sua presença e importância em quase todos os aspectos relacionados aos ambientes computacionais.

Para facilitar seus estudos, este material foi organizado em três unidades que buscam apresentar desde os princípios básicos relacionados aos sistemas operacionais e, consequentemente, aos sistemas operacionais distribuídos, até suas implementações.

Assim, sua caminhada iniciará na Unidade 1, na qual você encontrará os primeiros fundamentos sobre sistemas operacionais e sua evolução histórica, desde meados dos anos 1940 até os dias de hoje. Para compreender os fundamentos de um sistema operacional, você estudará não somente os conceitos básicos, mas também sua arquitetura básica, os tipos de sistemas operacionais existentes e como estes são capazes de gerenciar os processos de um sistema computacional.

O entendimento sobre sistemas operacionais é fundamental para o estudo sobre os sistemas distribuídos, assunto abordado na Unidade 2. De forma semelhante à Unidade 1, aqui você também será introduzido aos conceitos fundamentais sobre os sistemas distribuídos e sua arquitetura, os tipos de sistemas distribuídos e como os processos são executados. Mais adiante em seus estudos, você será apresentado aos conceitos de computação móvel e ubíqua, uma das grandes evoluções pelas quais passaram os sistemas distribuídos com o advento das redes de computadores e a popularização, nos dias de hoje, de dispositivos portáteis e móveis.

Após compreender o que são e como estão organizados os sistemas distribuídos, a Unidade 3 tratará dos aspectos relacionados à comunicação em ambientes distribuídos e aplicações distribuídas. Você perceberá que dada a característica distribuída deste tipo de sistema, conhecer os aspectos relacionados à comunicação é essencial. Estudará ainda aspectos críticos para o bom funcionamento da comunicação como consistência, tolerância a falhas e segurança. Por fim, você poderá aprender sobre implementações e aplicações com relação aos sistemas distribuídos.

Para que você possa assimilar ainda mais os conteúdos que lhe serão apresentados, ao longo do texto são destacados aspectos que merecem especial atenção, e ao final de cada tópico que compõe as unidades, você será convidado a realizar autoatividades para melhor fixação do seu aprendizado.

Desejamos que sua caminhada seja proveitosa!

Prof.^a Muriel de Fátima Bernhardt

Prof. Carlos Andre de Sousa Rocha



Você já me conhece das outras disciplinas? Não? É calouro? Enfim, tanto para você que está chegando agora à UNIASSELVI quanto para você que já é veterano, há novidades em nosso material.

Na Educação a Distância, o livro impresso, entregue a todos os acadêmicos desde 2005, é o material base da disciplina. A partir de 2017, nossos livros estão de visual novo, com um formato mais prático, que cabe na bolsa e facilita a leitura.

O conteúdo continua na íntegra, mas a estrutura interna foi aperfeiçoada com nova diagramação no texto, aproveitando ao máximo o espaço da página, o que também contribui para diminuir a extração de árvores para produção de folhas de papel, por exemplo.

Assim, a UNIASSELVI, preocupando-se com o impacto de nossas ações sobre o ambiente, apresenta também este livro no formato digital. Assim, você, acadêmico, tem a possibilidade de estudá-lo com versatilidade nas telas do celular, tablet ou computador.

Eu mesmo, UNI, ganhei um novo layout, você me verá frequentemente e surgirei para apresentar dicas de vídeos e outras fontes de conhecimento que complementam o assunto em questão.

Todos esses ajustes foram pensados a partir de relatos que recebemos nas pesquisas institucionais sobre os materiais impressos, para que você, nossa maior prioridade, possa continuar seus estudos com um material de qualidade.

Aproveito o momento para convidá-lo para um bate-papo sobre o Exame Nacional de Desempenho de Estudantes – ENADE.

Bons estudos!

BATE SOBRE O PAPO ENADE!



Olá, acadêmico!



Você já ouviu falar sobre o ENADE?

Se ainda não ouviu falar nada sobre o ENADE, agora você receberá algumas informações sobre o tema.

Ouviu falar? Ótimo, este informativo reforçará o que você já sabe e poderá lhe trazer novidades.



Vamos lá!



Qual é o significado da expressão ENADE?

EXAME NACIONAL DE DESEMPENHO DOS ESTUDANTES

Em algum momento de sua vida acadêmica você precisará fazer a prova ENADE.



Que prova é essa?



É **obrigatória**, organizada pelo INEP – Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira.

Quem determina que esta prova é obrigatória... O **MEC – Ministério da Educação**.

O objetivo do MEC com esta prova é o de avaliar seu desempenho acadêmico assim como a qualidade do seu curso.



Fique atento! Quem não participa da prova fica impedido de se formar e não pode retirar o diploma de conclusão do curso até regularizar sua situação junto ao MEC.



Não se preocupe porque a partir de hoje nós estaremos auxiliando você nesta caminhada.

Você receberá outros informativos como este, complementando as orientações e esclarecendo suas dúvidas.



Você tem uma trilha de aprendizagem do ENADE, receberá e-mails, SMS, seu tutor e os profissionais do polo também estarão orientados.

Participará de webconferências entre outras tantas atividades para que esteja preparado para #mandar bem na prova ENADE.

Nós aqui no NEAD e também a equipe no polo estamos com você para vencermos este desafio.

Conte sempre com a gente, para juntos mandarmos bem no ENADE!





Olá, acadêmico! Iniciamos agora mais uma disciplina e com ela um novo conhecimento.

Com o objetivo de enriquecer seu conhecimento, construímos, além do livro que está em suas mãos, uma rica trilha de aprendizagem, por meio dela você terá contato com o vídeo da disciplina, o objeto de aprendizagem, materiais complementares, entre outros, todos pensados e construídos na intenção de auxiliar seu crescimento.



Acesse o QR Code, que levará ao AVA, e veja as novidades que preparamos para seu estudo.

Conte conosco, estaremos juntos nesta caminhada!

SUMÁRIO

UNIDADE 1 – FUNDAMENTOS DE SISTEMAS OPERACIONAIS	1
TÓPICO 1 – CONCEITOS BÁSICOS SOBRE SISTEMAS OPERACIONAIS.....	3
1 INTRODUÇÃO	3
2 DEFININDO UM SISTEMA COMPUTACIONAL	3
3 O QUE É UM SISTEMA OPERACIONAL	6
4 FUNÇÕES DO SISTEMA OPERACIONAL.....	9
RESUMO DO TÓPICO 1.....	14
AUTOATIVIDADE	15
TÓPICO 2 – HISTÓRICO DE EVOLUÇÃO DOS SISTEMAS OPERACIONAIS	19
1 INTRODUÇÃO	19
2 BREVE HISTÓRICO SOBRE O SURGIMENTO DOS SISTEMAS OPERACIONAIS	19
3 DÉCADA DE 1940 OU PRIMEIRA GERAÇÃO	21
4 DÉCADA DE 1950 OU SEGUNDA GERAÇÃO	22
5 DÉCADAS DE 1960 E 1970 OU TERCEIRA GERAÇÃO	24
6 DÉCADAS DE 1980 E 1990 OU QUARTA GERAÇÃO	26
7 ANOS 2000 ATÉ O PRESENTE	29
RESUMO DO TÓPICO 2.....	30
AUTOATIVIDADE	31
TÓPICO 3 – ORGANIZAÇÃO E ARQUITETURA DE UM SISTEMA OPERACIONAL	35
1 INTRODUÇÃO	35
2 SERVIÇOS SUPORTADOS PELO SO	35
3 ARQUITETURAS SUPORTADAS PELO SO	40
3.1 ARQUITETURA MONOLÍTICA	40
3.2 ARQUITETURA EM CAMADAS.....	41
3.3 MÁQUINAS VIRTUAIS	42
3.4 ARQUITETURAS MICROKERNEL.....	44
RESUMO DO TÓPICO 3.....	45
AUTOATIVIDADE	46
TÓPICO 4 – TIPOS DE SISTEMAS OPERACIONAIS.....	49
1 INTRODUÇÃO	49
2 MONOTAREFA.....	50
3 MULTITAREFA	51
4 MÚLTIPLOS PROCESSADORES	54
RESUMO DO TÓPICO 4.....	59
AUTOATIVIDADE	60
TÓPICO 5 – GERENCIAMENTO DE PROCESSOS	63
1 INTRODUÇÃO	63
2 PROCESSOS	64
3 ESCALONAMENTO	68

4 MEMÓRIA E ARMAZENAMENTO	71
4.1 MEMÓRIA PRINCIPAL	71
4.2 MEMÓRIA VIRTUAL	72
4.3 ESTRUTURA DE ARMAZENAMENTO.....	74
LEITURA COMPLEMENTAR.....	76
RESUMO DO TÓPICO 5.....	78
AUTOATIVIDADE	79
UNIDADE 2 – SISTEMAS DISTRIBUÍDOS	83
TÓPICO 1 – CONCEITOS BÁSICOS SOBRE SISTEMAS DISTRIBUÍDOS	85
1 INTRODUÇÃO	85
2 O QUE SÃO SISTEMAS DISTRIBUÍDOS?.....	88
3 PROPÓSITO	90
4 EXEMPLOS DE SISTEMAS DISTRIBUÍDOS.....	96
RESUMO DO TÓPICO 1.....	97
AUTOATIVIDADE	98
TÓPICO 2 – ARQUITETURA PARA SISTEMAS DISTRIBUÍDOS.....	101
1 INTRODUÇÃO	101
2 ARQUITETURAS FÍSICAS	101
3 ARQUITETURAS PARA SISTEMAS DISTRIBUÍDOS	103
3.1 ELEMENTOS ARQUITETÔNICOS	104
3.2 PADRÕES ARQUITETÔNICOS	105
3.3 SOLUÇÕES DE MIDDLEWARE ASSOCIADAS.....	108
4 ARQUITETURAS FUNDAMENTAIS	108
RESUMO DO TÓPICO 2.....	111
AUTOATIVIDADE	112
TÓPICO 3 – PROCESSOS.....	117
1 INTRODUÇÃO	117
2 THREADS	117
3 VIRTUALIZAÇÃO	121
4 MIGRAÇÃO DE CÓDIGO	125
RESUMO DO TÓPICO 3.....	128
AUTOATIVIDADE	129
TÓPICO 4 – TIPOS DE SISTEMAS DISTRIBUÍDOS	133
1 INTRODUÇÃO	133
2 SISTEMAS DE COMPUTAÇÃO DISTRIBUÍDOS	133
2.1 CLUSTER	134
2.2 GRID (GRADE)	139
3 SISTEMAS DE INFORMAÇÃO DISTRIBUÍDOS	141
RESUMO DO TÓPICO 4.....	145
AUTOATIVIDADE	146
TÓPICO 5 – COMPUTAÇÃO MÓVEL E UBÍQUA	149
1 INTRODUÇÃO	149
2 COMPUTAÇÃO MÓVEL.....	151
3 COMPUTAÇÃO PERVASIVA.....	154

4 COMPUTAÇÃO UBÍQUA	155
5 SEGURANÇA	157
LEITURA COMPLEMENTAR.....	161
RESUMO DO TÓPICO 5.....	162
AUTOATIVIDADE	163
UNIDADE 3 – COMUNICAÇÃO EM AMBIENTES DISTRIBUÍDOS E APLICAÇÕES DISTRIBUÍDAS	167
TÓPICO 1 – COMUNICAÇÃO.....	169
1 INTRODUÇÃO	169
2 REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS	170
2.1 PROTOCOLOS DE COMUNICAÇÃO.....	172
2.2 ATRIBUTOS DAS REDES PARA OS SISTEMAS DISTRIBUÍDOS	176
3 CARACTERÍSTICAS GERAIS DA COMUNICAÇÃO ENTRE PROCESSOS EM SISTEMAS DISTRIBUÍDOS	178
3.1 CHAMADA DE PROCEDIMENTO REMOTO.....	180
3.2 MIDDLEWARE ORIENTADO A MENSAGEM.....	183
3.3 FLUXO DE DADOS.....	184
3.4 MULTICASTING	185
RESUMO DO TÓPICO 1.....	186
AUTOATIVIDADE	188
TÓPICO 2 – CONSISTÊNCIA E TOLERÂNCIA A FALHAS.....	193
1 INTRODUÇÃO	193
2 REPLICAÇÃO E CONSISTÊNCIA	193
2.1 MODELOS DE CONSISTÊNCIA	196
3 TOLERÂNCIA A FALHAS	199
3.1 SISTEMAS CONFIÁVEIS	200
3.2 TIPOS DE FALHAS	201
3.3 DETECÇÃO E MASCARAMENTO DE FALHAS	202
3.4 RECUPERAÇÃO DE ERROS	203
RESUMO DO TÓPICO 2.....	204
AUTOATIVIDADE	206
TÓPICO 3 – SEGURANÇA.....	209
1 INTRODUÇÃO	209
2 TÉCNICAS DE SEGURANÇA.....	212
3 SEGURANÇA NOS CANAIS DE COMUNICAÇÃO	218
RESUMO DO TÓPICO 3.....	219
AUTOATIVIDADE	220
TÓPICO 4 – IMPLEMENTAÇÕES DE SISTEMAS E APLICAÇÕES DISTRIBUÍDAS.....	225
1 INTRODUÇÃO	225
2 SISTEMAS DISTRIBUÍDOS BASEADOS EM OBJETOS	225
2.1 ARQUITETURA	226
2.2 PROCESSOS	228
2.3 COMUNICAÇÃO.....	228
2.4 CONSISTÊNCIA E REPLICAÇÃO	229
2.5 TOLERÂNCIA A FALHAS.....	229
2.6 SEGURANÇA.....	229

3 SISTEMAS DE ARQUIVOS DISTRIBUÍDOS	230
3.1 ARQUITETURA.....	230
3.2 PROCESSOS	232
3.3 COMUNICAÇÃO.....	233
3.4 CONSISTÊNCIA E REPLICAÇÃO	233
3.5 TOLERÂNCIA A FALHAS.....	234
3.6 SEGURANÇA.....	235
4 SISTEMAS DISTRIBUÍDOS BASEADOS NA WEB	236
4.1 ARQUITETURA.....	237
4.2 PROCESSOS	238
4.3 COMUNICAÇÃO.....	239
4.4 CONSISTÊNCIA E REPLICAÇÃO	239
4.5 TOLERÂNCIA A FALHAS.....	240
4.6 SEGURANÇA.....	240
5 SISTEMAS MULTIMÍDIA DISTRIBUÍDOS.....	241
5.1 DADOS MULTIMÍDIA	242
5.2 GERENCIAMENTO DE QUALIDADE DE SERVIÇO.....	243
5.3 GERENCIAMENTO DE RECURSOS	244
LEITURA COMPLEMENTAR.....	246
RESUMO DO TÓPICO 4.....	248
AUTOATIVIDADE	250
REFERÊNCIAS	253

UNIDADE 1

FUNDAMENTOS DE SISTEMAS OPERACIONAIS

OBJETIVOS DE APRENDIZAGEM

A partir do estudo desta unidade, você deverá ser capaz de:

- compreender os fundamentos básicos sobre os sistemas operacionais;
- conhecer o histórico de evolução dos sistemas operacionais relacionando-os aos sistemas computacionais;
- conhecer e compreender como está organizado um sistema operacional em termos de sua arquitetura;
- identificar os diferentes tipos de sistemas operacionais, seus usos e funcionalidades.

PLANO DE ESTUDOS

Esta unidade está dividida em cinco tópicos. No decorrer da unidade você encontrará autoatividades com o objetivo de reforçar o conteúdo apresentado.

TÓPICO 1 – CONCEITOS BÁSICOS SOBRE SISTEMAS OPERACIONAIS

TÓPICO 2 – HISTÓRICO DE EVOLUÇÃO DOS SISTEMAS OPERACIONAIS

TÓPICO 3 – ORGANIZAÇÃO E ARQUITETURA DE UM SISTEMA OPERACIONAL

TÓPICO 4 – TIPOS DE SISTEMAS OPERACIONAIS

TÓPICO 5 – GERENCIAMENTO DE PROCESSOS



Preparado para ampliar seus conhecimentos? Respire e vamos em frente! Procure um ambiente que facilite a concentração, assim absorverá melhor as informações.

CONCEITOS BÁSICOS SOBRE SISTEMAS OPERACIONAIS

1 INTRODUÇÃO

Para que você possa iniciar seus estudos sobre Sistemas e Aplicações Distribuídas é importante que seu ponto de partida esteja em um conceito anterior: o conceito de Sistema Operacional.

Entender o que é e como surgiram e evoluíram os Sistemas Operacionais, como o Windows, que talvez esteja sendo executado em seu computador agora, por exemplo, permitirá a você compreender não somente como os sistemas computacionais são construídos, mas como se dá seu funcionamento.

Perceba então que, o funcionamento de seu computador, notebook, ou mesmo smartphone só é possível graças a esta categoria muito específica de software denominada Sistema Operacional, sobre o qual você conhecerá mais detalhes na sequência de sua leitura.

2 DEFININDO UM SISTEMA COMPUTACIONAL

Quando falamos em computador logo nos vem à mente uma máquina capaz de realizar uma série de funções ou atividades que auxiliam em nossa vida cotidiana, seja no trabalho, na escola ou mesmo em momentos de lazer. Nos dias de hoje, diferentemente de quinze ou vinte anos atrás, as pessoas compreendem que um computador é formado por dois elementos principais: seu hardware e seus softwares.

Você certamente já deve ter ouvido aquela brincadeira que distingue esses dois elementos assim “hardware é aquilo que a gente chuta, software é com o que a gente briga!”.

Então, é importante compreender que o hardware está relacionado aos componentes físicos de um computador, a máquina em si. Por outro lado, o software compreende, falando muito diretamente, os programas que serão executados dentro deste hardware.

Então, o hardware de um computador torna-se realmente útil quando executa um software. “Sem software, um computador é basicamente um monte inútil de metal” (TANENBAUM; WOODHULL, 2008, p. 21).

O papel do software em um computador está relacionado não somente a fazê-lo funcionar gerenciando seus recursos, mas a permitir que o usuário execute ações de acordo com sua necessidade. Em outras palavras, o software permitirá ao seu computador guardar e recuperar informações, como seus trabalhos de aula ou suas fotos; reproduzir músicas e vídeos, fazer pesquisas na internet e enviar ou receber seus e-mails, além de muitas outras atividades.

Imaginamos que você tenha percebido que, ao falarmos do software do computador, atribuímos a ele duas ações distintas: primeiro, fazer o computador funcionar e, segundo, permitir que o usuário faça o que desejar. Ao perceber isso, fica mais fácil entender que o software que é executado por um computador pode ser de dois tipos: podemos estar falando de um software de sistema, ou podemos estar falando de um software aplicativo.

Veja como cada um pode ser definido:

- **Software de sistema:** “[...] programas de sistema, que gerenciam a operação do computador em si [...]” (TANENBAUM; WOODHULL, 2008, p. 21).
- **Software aplicativo:** “[...] realizam o trabalho real desejado pelo usuário” (TANENBAUM; WOODHULL, 2008, p. 21).



Um software de sistema funciona como um intermediário entre o hardware do computador e os vários softwares aplicativos instalados.

A Figura 1 mostra como funciona esta relação entre os dois tipos de software e o hardware do computador.

FIGURA 1 – SOFTWARE DE SISTEMA, SOFTWARES APLICATIVOS E HARDWARE



FONTE: Adaptado de Turban, Rainer Jr. e Potter (2003)

Agora que você já distingue os elementos de um computador em hardware e software, e que este último pode ser de dois tipos, chegou a hora de compreender um novo termo, mais abrangente, mas que será utilizado, daqui para frente, sempre que quisermos nos referir ao computador. Estamos falando dos **sistemas computacionais**.

Um sistema computacional vai um pouco além na explicação nos conceitos de hardware e software e considera, além destes elementos mais um: o usuário. Então, quando estivermos falando de um sistema computacional estamos nos referindo, basicamente a quatro componentes: hardware, softwares aplicativos, software de sistema (que você já conhece) e o usuário, ou seja, as pessoas que vão utilizar o computador, como você, por exemplo! (SILBERSCHATZ; GALVIN; GAGNE, 2015).

Ao referir-se a um sistema computacional, é importante também compreender que seu hardware normalmente “[...] consiste em um ou mais processadores, memória principal, discos, impressoras, teclado, tela, interfaces de rede e outros dispositivos de entrada/saída” (TANENBAUM; WOODHULL, 2008, p. 21).



As operações de entrada/saída são normalmente identificadas pelas suas iniciais, E/S, ou também pelo termo I/O, do inglês input/output.

Veja como cada um dos elementos de um sistema computacional se relaciona na Figura 2:

FIGURA 2 – SISTEMA COMPUTACIONAL



FONTE: Adaptado de Machado e Maia (2017)

Agora que você já comprehende todos os elementos envolvidos em um sistema computacional, exatamente como este que você está utilizando agora para ler este material, vamos nos dedicar a conhecer, com mais detalhes, o principal software de sistema presente em um sistema computacional. Este software de sistema é muito conhecido pelo nome de Sistema Operacional e, apesar de em um primeiro momento o nome não lhe parecer muito familiar, logo você vai perceber que o Sistema Operacional já é um velho conhecido seu.



Para ficar mais fácil, daqui em diante indicaremos o nome Sistema Operacional por suas iniciais: SO.

3 O QUE É UM SISTEMA OPERACIONAL

Como você acabou de aprender, um SO é um software de sistema, e, como tal, é representado como uma camada que existe exatamente para servir de interface entre o hardware e os programas aplicativos utilizados pelo usuário em um sistema computacional (OLIVEIRA; CARISSIMI; TOSCANI, 2010; SILBERSCHATZ; GALVIN; GAGNE, 2015).



"Interface com o usuário é um conjunto de recursos de software que fornece aos usuários controle direto sobre os objetos visíveis na tela. Assim, os usuários podem usar os ícones, menus [...], janelas [...] em vez de uma sintaxe de comandos complexa" (TURBAN; RAINER JR.; POTTER, 2003, p. 22).

A existência do SO, como uma camada intermediária entre o hardware e o usuário e seus aplicativos, é importante porque “o hardware deve fornecer mecanismos apropriados para assegurar a operação correta do sistema de computação e impedir que programas de usuários interfiram na operação adequada do sistema” (SILBERSCHATZ; GALVIN; GAGNE, 2015, p. 15). Com esse entendimento, podemos então apresentar uma definição mais completa para um SO:

Um sistema operacional, por mais complexo que possa parecer, é apenas um conjunto de rotinas executado pelo processador, de forma semelhante aos programas dos usuários. Sua principal função é controlar o funcionamento de um computador, gerenciando a utilização e o compartilhamento dos seus diversos recursos, como processadores, memórias e dispositivos de entrada e saída (MACHADO; MAIA, 2011; 2017, p. 3).

Repare que, anteriormente, ao definir o SO, também introduzimos uma outra questão importantíssima: se já sabemos o que é um SO, agora é importante entender ainda qual sua finalidade, ou seja, qual o objetivo de um SO em um sistema computacional.

Em relação ao seu objetivo ou finalidade, um SO deve, como já comentado antes “[...] funcionar como uma interface entre o usuário e o computador, tornando sua utilização mais simples, rápida e segura” (MACHADO; MAIA, 2011; 2017, p. 1).

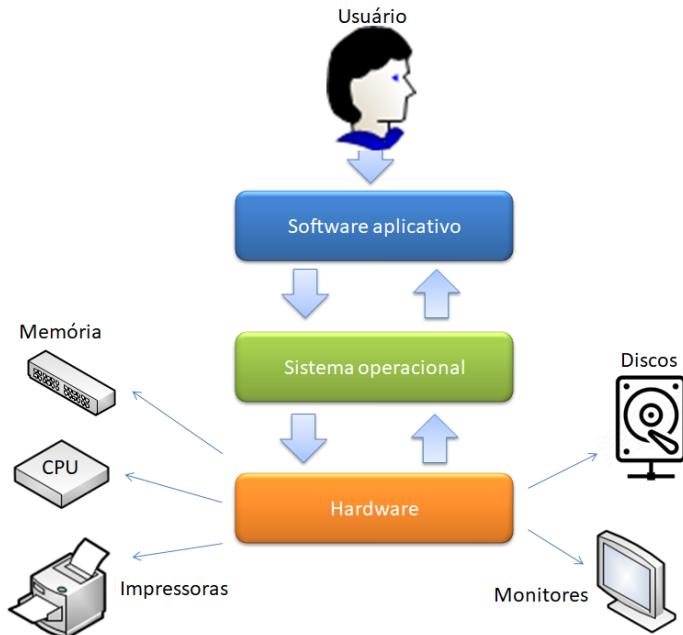
Outro objetivo importante do SO está relacionado a permitir acesso aos diversos periféricos do sistema computacional. Assim, sempre que uma operação de E/S é executada pelo usuário, esta solicita permissão ao SO. Por exemplo, quando você encaminha um documento feito no editor de texto para a impressora, uma solicitação de impressão é enviada ao SO que permite ou não que a ação seja completada (OLIVEIRA; CARISSIMI; TOSCANI, 2010).



Periféricos são dispositivos externos ao hardware do computador que servem para a realização de entradas de dados pelo usuário e saídas de dados pelo computador. Por isso mesmo são conhecidos como dispositivos de entrada (periféricos como teclado, mouse, joysticks) e dispositivos de saída (periféricos como monitores e impressoras).

Lembre-se que o hardware dos computadores modernos é composto de uma grande variedade de dispositivos como processador, memória, discos, impressoras. Daí a importância do SO garantir que cada um desses dispositivos será alocado de forma organizada pelos vários softwares aplicativos do usuário (TANENBAUM; WOODHULL, 2008). Veja como este objetivo gerencial do SO é atendido na Figura 3.

FIGURA 3 – OBJETIVO DO SO



FONTE: Adaptado de Machado e Maia (2017)

A partir de uma visão mais prática e menos tecnicista, pode-se afirmar que o objetivo do SO é tornar a vida do usuário mais fácil!

O sistema operacional procura tornar a utilização do computador, ao mesmo tempo, mais eficiente e mais conveniente. A utilização mais eficiente busca um maior retorno no investimento feito no hardware. Maior eficiência significa mais trabalho obtido do mesmo hardware. Uma utilização mais conveniente vai diminuir o tempo necessário para a construção dos programas. Isso também implica a redução no custo do software, pois são necessárias menos horas de programador (OLIVEIRA; CARISSIMI; TOSCANI, 2010, p. 23).

A eficiência e a conveniência, ou seja, tirar o máximo proveito possível do sistema computacional em sua utilização depende da finalidade do SO, ou seja, para que tipo de sistema computacional ele foi desenvolvido. Existe uma variedade de sistemas computacionais, desde os computadores de grande porte como os *mainframes*, passando pelos computadores pessoais (como aqueles que existem nas empresas e nos laboratórios de informática das escolas), até os computadores móveis (como os notebooks e smartphones). Em cada caso há um tipo específico de SO sendo executado. Veja no Quadro 1:

QUADRO 1 – TIPOS DE SISTEMAS COMPUTACIONAIS E OS DIFERENTES OBJETIVOS DE SO

Tipo de sistema computacional	Objetivo do SO
Mainframe	Otimizar a utilização do hardware.
Computadores pessoais (PCs)	Suportar jogos complexos, aplicações comerciais e mais uma variedade de aplicativos.
Computadores móveis	Fornecer um ambiente no qual o usuário pode interagir facilmente com o computador para executar programas.

FONTE: Adaptado de Silberschatz, Galvin e Gagne (2015)



Considerados um tipo de servidor, os *mainframes* são sistemas computacionais de grande porte, como recursos de hardware e software direcionados. São populares em grandes empresas por executarem aplicações que podem ser acessadas por milhares de usuários. Exemplos de aplicações para *mainframes* envolvem sistemas de reservas de passagens aéreas (TURBAN; RAINER JR.; POTTER, 2003).

4 FUNÇÕES DO SISTEMA OPERACIONAL

Você já percebeu que, sempre que aprendemos algo novo ou nos aprofundamos mais sobre algum assunto que já conhecíamos, nos perguntamos: "o que é?", "para que serve?" e "como funciona?". É exatamente isso que estamos fazendo sobre o SO! Respondendo à essas perguntas que são tão naturais no nosso processo de aprendizado.

Então, agora que você já entendeu o conceito deste software de sistema chamado SO (respondendo à pergunta "o que é?") e seus objetivos dentro de um sistema computacional (respondendo à pergunta "para que serve?"), vamos seguindo adiante! Vamos agora responder à terceira pergunta ("como funciona?") e identificar as funções de um SO. Sobre as funções de um SO podemos dizer o seguinte:

Para atingir os objetivos propostos, o sistema operacional oferece diversos tipos de serviços. A definição precisa dos serviços depende do sistema operacional em consideração. Entretanto, a maioria dos sistemas operacionais oferece um conjunto básico de serviços, sempre necessários (OLIVEIRA; CARISSIMI; TOSCANI, 2010, p. 23).



Funções, tarefas ou serviços são termos sinônimos utilizados para definir como um SO trabalha. O termo depende do autor que você estiver pesquisando. Para facilitar seu entendimento, deste ponto em diante, sempre utilizaremos o termo "função".

Como já comentado antes, um SO em um sistema computacional deve ser tanto "eficiente" quanto "conveniente". A conveniência vem do papel inicial do SO de servir como interface para o usuário e o sistema computacional, permitindo que as operações complexas que são realizadas entre ele e o hardware fiquem "transparentes", ou seja, as coisas acontecem, mas o usuário não precisa visualizar como acontecem.

Algumas dessas funções estão listadas no Quadro 2:

QUADRO 2 – FUNÇÕES COMUNS DO SO

Funções comuns do SO	
Monitorar o desempenho.	Criar e manter diretórios.
Corrigir erros.	Formatar dispositivos como pen drives.
Fornecer e manter a interface do usuário.	Controlar o monitor do computador.
Inicializar o computador.	Enviar documentos para a impressora.
Ler os programas para a memória.	Preservar a segurança e limitar o acesso.
Gerenciar a alocação de memória para esses programas.	Gerenciar o envio/recebimento de mensagens instantâneas por SMS ou aplicativos. Localizar arquivos.
Colocar os arquivos e programas no armazenamento secundário.	Detectar vírus. Compactar dados.

FONTE: Adaptado de Turban, Rainer Jr. e Potter (2003)

Dentre as funções que um SO deve ser capaz de realizar — listadas no Quadro 2 —, algumas são consideradas complexas e merecem ser detalhadas aqui para seu conhecimento. Tais funções estão relacionadas à supervisão e operação geral do computador [...] incluindo a monitoração do status do computador e o escalonamento de operações, o que inclui os processos de entrada e saída” (TURBAN; RAINER JR.; POTTER, 2003, p. 101).

Veja na sequência a lista de algumas funções complexas que merecem sua atenção:

- Gerenciamento de processos.
- Multiprocessamento.
- Memória virtual.
- Gerenciamento de arquivos.
- Segurança.
- Facilidade de acesso.
- Tolerância a falhas.

Gerenciamento de processos é a função exercida pelo SO para controlar os programas (softwares aplicativos, por exemplo) que estão sendo executados em algum momento pelo processador.



Você sabia que quando um programa está sendo executado pelo processador ele é identificado pelo SO como um *job*?

Poderíamos descrever um cenário simples em que você está utilizando um software aplicativo para digitar suas atividades como o Word, por exemplo: um programa está sendo executado pelo processador. Em um cenário mais complicado, além de digitar suas atividades você também está pesquisando páginas no seu navegador e executando um programa para ouvir músicas: três programas estão sendo executados. Isso é gerenciamento de processos! Avaliando este exemplo, fica fácil entender porque esta função tornou-se uma das mais importantes do SO.

Multiprocessamento é caracterizado pela existência de dois ou mais processadores interligados executando programas distintos ou executando, simultaneamente, um mesmo programa. “Existem inúmeras vantagens em sistemas com múltiplos processadores, como desempenho, escalabilidade, relação custo/desempenho, tolerância a falhas, disponibilidade e balanceamento de carga” (MACHADO; MAIA, 2017, p. 221).



“Escalabilidade é a capacidade de adicionar novos processadores ao hardware do sistema” (MACHADO; MAIA, 2017, p. 221).



"Balanceamento de carga é a distribuição do processamento entre os diversos componentes da configuração, a partir da carga de cada processador, melhorando, assim, o desempenho do sistema como um todo" (MACHADO; MAIA, 2017, p. 222).

Memória virtual "[...] é uma técnica sofisticada e poderosa de gerencia de memória, em que as memórias principal e secundária são combinadas dando ao usuário a ilusão de existir uma memória muito maior que a capacidade real da memória principal" (MACHADO; MAIA, 2017, p. 159). Em outras palavras, tem-se a sensação que o computador possui mais memória do que de fato existe. Essa simulação de uma quantidade maior de memória, entre outras coisas, aumenta a velocidade do seu computador além de executar com mais eficiência certos tipos de programas.

Gerenciamento de arquivos está entre as funções essenciais do SO, pois está relacionada à capacidade de um processo de armazenar e recuperar grandes volumes de informações e a maneira pela qual o SO organiza estas informações em estruturas chamadas arquivos (MACHADO; MAIA, 2011; 2017).

Os arquivos são gerenciados pelo sistema operacional de maneira a facilitar o acesso dos usuários ao seu conteúdo. A parte do sistema responsável por essa gerência é denominada sistema de arquivos. O sistema de arquivos é a parte mais visível de um sistema operacional, pois a manipulação de arquivos é uma atividade frequentemente realizada pelos usuários, devendo sempre ocorrer de maneira uniforme, independente dos diferentes dispositivos de armazenamento (MACHADO; MAIA, 2017, p. 194).



"Um arquivo é constituído por informações logicamente relacionadas. Essas informações podem representar instruções ou dados. Um arquivo executável, por exemplo, contém instruções compreendidas pelo processador, enquanto um arquivo de dados pode ser estruturado livremente como um arquivo-texto [...]" (MACHADO; MAIA, 2017, p. 194).

Não se pode falar de tantas funções e deixar de lado uma tarefa igualmente importante como a **Segurança**. Como você já deve ter visto no ambiente "Este computador" do Windows, o gerenciamento dos arquivos em seu computador é feito a partir de uma estrutura de diretórios. Dentro desta estrutura organizada e mantida pelo SO você pode criar, recuperar, ler, alterar e até eliminar arquivos. Por isso é importante controlar o acesso a esses arquivos (TURBAN; RAINER JR.; POTTER, 2003).

É comum que o ambiente onde esses arquivos estejam compartilhados seja por mais de um usuário. Por exemplo, em sua casa, além de você, quantas pessoas podem acessar seu computador? Talvez mais uma ou duas. Então é importantíssimo que o SO disponha de mecanismos para proteger estes arquivos para que não sejam danificados pela ação de outros usuários, mecanismos como controle de acesso ou compartilhamento a partir de permissões.

Outra importante função do SO é promover a **facilidade de acesso** aos recursos do sistema. O acesso a recursos, como monitores, impressoras, unidades de CD/DVD (*Compact Disc/Digital Video Disc*), deve ocorrer de forma transparente para o usuário, permitindo que ele execute suas tarefas sem se preocupar como a comunicação entre o computador e a impressora, por exemplo, está acontecendo. Esta mesma facilidade de acesso deve ser garantida pelo SO em situações de compartilhamento de um mesmo recurso por vários usuários simultaneamente, garantindo “[...] uma distribuição justa e eficiente dos recursos” (OLIVEIRA; CARISSIMI, TOSCANI, 2010, p. 23).



"Não é apenas em sistemas multiusuário que o sistema operacional é importante.

Se pensarmos que um computador pessoal nos permite executar diversas tarefas ao mesmo tempo, como imprimir um documento, copiar um arquivo pela Internet ou processar uma planilha, o sistema operacional deve ser capaz de controlar a execução concorrente de todas essas atividades" (MACHADO; MAIA, 2011; 2017, p. 5).

Finalmente, a **Tolerância a falhas** “[...] é a capacidade de manter o sistema em operação mesmo em casos de falha em algum componente” (MACHADO; MAIA, 2017, p. 222). A capacidade do SO de ser tolerante à falhas protege o sistema contra falhas ou erros ocasionados pelo hardware (por exemplo, uma situação de queda de energia que desligaria o computador forçadamente) ou pelo próprio usuário (por exemplo, quando o usuário está copiando arquivos de um pen drive para o computador, e antes que esta ação termine, ele o remove) e mesmo assim continuar funcionando (TURBAN; RAINER JR.; POTTER, 2003).



Você poderá encontrar mais sobre o conceito dos sistemas operacionais, em especial sobre a arquitetura de computadores pessoais acessando o seguinte material:

BEZERRA, R. M. **Conceitos de sistemas operacionais**. Disponível em: http://www2.ufba.br/~romildo/downloads/ifba/inf400_conceitos_so.pdf. Acesso em: 4 jul. 2019.

RESUMO DO TÓPICO 1

Neste tópico, você aprendeu que:

- Um sistema computacional é formado pelo hardware do computador, seus dois tipos de software, de sistema e aplicativo, e o usuário.
- O Sistema Operacional (SO) é um software de sistema e funciona como uma camada intermediária entre o hardware e os softwares aplicativos utilizados pelo usuário.
- O SO deve garantir eficiência e conveniência ao usuário, ou seja, tirar o máximo proveito do hardware do computador e redução do tempo para a construção e execução de programas.
- O SO realiza funções básicas e complexas ao servir como interface entre o hardware e usuário, como, por exemplo, localizar arquivos, corrigir erros, acessar e formatar dispositivos como pen drives, gerenciar o envio/recebimento de mensagens, gerenciar processos e arquivos, garantir a facilidade de acesso e a tolerância a falhas.

AUTOATIVIDADE



- 1 Mencionou-se, neste tópico, que, ao software de um computador, são atribuídas duas ações distintas: primeiro, fazer o computador funcionar e segundo, permitir que o usuário faça o que desejar, fazendo com que identifiquemos dois tipos de software: o software de sistema e o software aplicativo. A partir disso, assinale a alternativa que estabelece uma definição para o software de sistema:
- a) () programas de sistema, que gerenciam a operação do computador em si.
 - b) () realizam o trabalho real desejado pelo usuário.
 - c) () consiste em um ou mais processadores, memória principal, discos, impressoras, teclado, tela, interfaces de rede e outros dispositivos de entrada/saída.
 - d) () pode ser entendido como um programa em execução.
- 2 Você viu que o principal objetivo do SO é “[...] funcionar como uma interface entre o usuário e o computador, tornando sua utilização mais simples, rápida e segura” (MACHADO; MAIA, 2011; 2017, p. 1). Esse objetivo é cumprido porque o SO executa uma série de funções como, por exemplo, monitorar o desempenho do computador, corrigir eventuais erros na execução de softwares aplicativos, fornecer e manter a interface do usuário e inicializar o computador. Há ainda outras funções, consideradas mais complexas, e que merecem nossa atenção. Nas assertivas a seguir são apresentadas as definições de algumas destas funções:
- I- Gerenciamento de processos é caracterizado pela existência de dois ou mais processadores interligados executando programas distintos ou executando, simultaneamente, um mesmo programa.
 - II- Multiprocessamento está entre as funções essenciais do SO, pois está relacionada à capacidade de um processo de armazenar e recuperar grandes volumes de informações.
 - III- Tolerância a falhas é a capacidade de manter o sistema em operação mesmo em casos de falha em algum componente.
 - IV- Gerenciamento de arquivos é a função exercida pelo SO para controlar os programas (softwares aplicativos, por exemplo) que estão sendo executados em algum momento pelo processador.
- A partir das definições fornecidas, estão CORRETAS as assertivas:
- a) () I e II.
 - b) () I e IV.
 - c) () Somente a III.
 - d) () Somente a I.

- 3 Um SO pode permitir que um computador execute vários programas na memória, ao mesmo tempo ou de maneira concorrente, com mostra a figura. Logo, o SO deve assegurar que todos executem sem interferência entre si. Isto é, sem que cada um dos programas afete de algum modo a corretude do comportamento individual um do outro (TANENBAUM; STEEN, 2007). A afirmação em destaque refere-se a qual função do SO:

Memória principal	Endereços
Sistema operacional (256 Kbytes)	00000 H
Programa usuário 1 (160 Kbytes)	3FFFF H 40000 H
Programa usuário 2 (64 Kbytes)	67FFF H 68000 H
Programa usuário 3 (32 Kbytes)	77FFF H 78000 H 7FFFF H

FONTE: Oliveira, Carissimi e Toscani (2010, p. 38)

- a) () Monitorar o desempenho.
 - b) () Formatar dispositivos como pen drives.
 - c) () Inicializar o computador.
 - d) () Gerenciar a alocação de memória para esses programas.
- 4 “[...] funcionar como uma interface entre o usuário e o computador, tornando sua utilização mais simples, rápida e segura” (MACHADO; MAIA, 2011; 2017, p. 1), trata-se do principal objetivo de um SO. Como parte desse objetivo, tem-se a detecção de vírus, localização, criação e exclusão de arquivos, criação e manutenção de pastas, existindo, ainda, outras funções. Considerando as assertivas a seguir, indique aquela que não representa a função de um SO:
- a) () Permitir que as aplicações recuperem/armazenem dados de/para a memória.
 - b) () Gerenciar o tráfego de dados entre os periféricos de um computador.
 - c) () Fornecer informações relevantes aos usuários quando da ocorrência de um erro no sistema.
 - d) () Realizar buscas/armazenamentos de dados em base de dados de softwares aplicativos.
- 5 Um SO pode permitir que um computador execute vários programas na memória, ao mesmo tempo ou de maneira concorrente, com mostra a figura. Logo, o SO deve assegurar que todos executem sem interferência entre si. Isto é, sem que cada um dos programas afete de algum modo a corretude do comportamento individual um do outro (TANENBAUM; STEEN, 2007). A afirmação em destaque refere-se as funções do SO:

Memória principal	Endereços
Sistema operacional (256 Kbytes)	00000 H
Programa usuário 1 (160 Kbytes)	3FFF H
Programa usuário 2 (64 Kbytes)	40000 H
Programa usuário 3 (32 Kbytes)	67FFF H
	68000 H
	77FFF H
	78000 H
	7FFFF H

FONTE: Oliveira, Carissimi e Toscani (2010, p. 38)

- I- Criar e manter diretórios.
- II- Preservar a segurança e limitar o acesso.
- III- Detectar vírus.
- IV- Ler os programas para a memória.

A partir das definições fornecidas, estão CORRETAS as assertivas:

- a) () I e III.
- b) () II e IV.
- c) () Somente a III.
- d) () Somente a I.

HISTÓRICO DE EVOLUÇÃO DOS SISTEMAS OPERACIONAIS

1 INTRODUÇÃO

Agora que você já foi devidamente apresentado ao que chamamos de sistema computacional ao SO podemos dar continuidade aos estudos. Então, neste tópico, você será convidado a viajar atrás no tempo para conhecer a origem dos SO.

Perceba que este retorno no tempo é importante porque permitirá que você estabeleça não somente uma comparação entre como os primeiros sistemas computacionais foram construídos em relação aos atuais, mas especialmente, poderá construir uma relação entre como funcionavam os primeiros SO e como funcionam hoje.

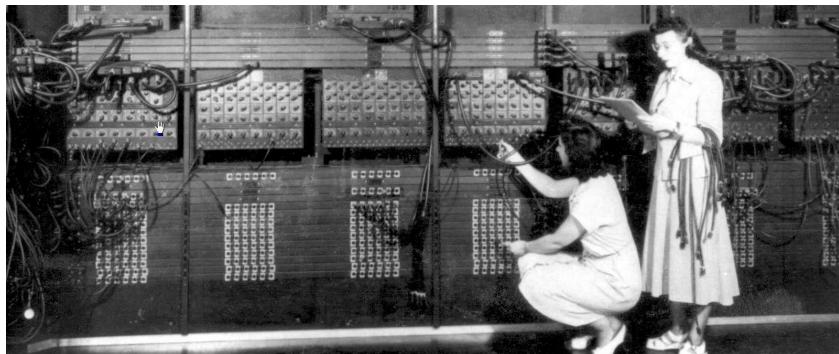
Dentre outras coisas, a evolução dos SO ao longo de meados do século XX, permitiu que pudéssemos executar várias tarefas, ou processos, ao mesmo tempo em um computador, ou ainda, que várias pessoas pudessem compartilhar uma impressora.

2 BREVE HISTÓRICO SOBRE O SURGIMENTO DOS SISTEMAS OPERACIONAIS

Desde o surgimento dos primeiros computadores, a evolução ocorrida com o hardware foi acompanhada pela evolução do software, especialmente do SO. Isso aconteceu exatamente pelas razões que você viu até chegar aqui: o que é o SO, seu objetivo e suas funções dentro de um sistema computacional. Desta forma, não há como falar da evolução histórica dos SO sem comentar como o hardware dos computadores também evoluiu (TANENBAUM; WOODHULL, 2008). E é exatamente sobre isso que você estudará a partir de agora.

Para começo de conversa, é importante que você entenda: por que razão o SO surgiu? “Nos primeiros computadores, a programação era realizada em linguagem de máquina, em painéis através de fios, exigindo, consequentemente, um grande conhecimento da arquitetura do hardware. Isso era uma grande dificuldade para os programadores da época” (MACHADO; MAIA, 2017, p. 3). Veja como era fazer a programação de um computador na Figura 4.

FIGURA 4 – PRIMEIROS COMPUTADORES

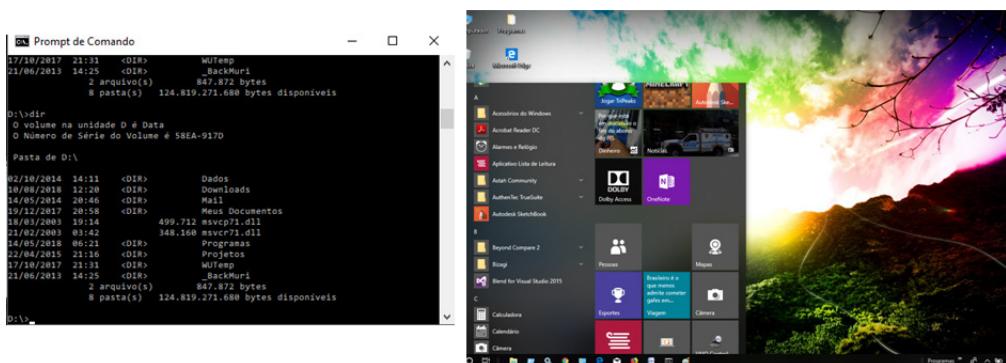


FONTE: <<https://1hzoda29f77r1yh9c33lm1ae-wpengine.netdna-ssl.com/wp-content/uploads/2015/03/eniac1-1024x432.jpg>>. Acesso em: 5 jul. 2019.

Então, a resposta a nossa pergunta anterior é simples e direta: para resolver este problema de interação entre os usuários e o computador era necessária alguma coisa que promovesse uma interface “simples, confiável e eficiente”. Em outras palavras, era importantíssimo que o hardware do computador não representasse um problema para fazê-lo funcionar. E é exatamente esse problema que o SO resolve! (MACHADO; MAIA, 2011; 2017).

Ao longo dos anos, os SO evoluíram para facilitar o uso dos computadores. Foi isso que permitiu a evolução de interfaces baseadas em comandos (bastante difíceis de utilizar) para interfaces gráficas (bem mais amigáveis), baseadas em menus e ícones manipulados a partir de um mouse. “Tudo isso para simplificar a utilização do computador” (OLIVEIRA; CARISSIMI, TOSCANI, 2010, p. 33). Veja melhor esta evolução do SO desde uma interface baseada em comandos (à esquerda) e uma interface gráfica (à direita) na Figura 5.

FIGURA 5 – COMPARAÇÃO ENTRE AS INTERFACES DE UM SO



FONTE: Os autores

Esta evolução do uso e da interface dos SO não aconteceu de uma hora para outra, mas vem acontecendo há mais de 50 anos. Essas mudanças são tão importantes e foram tão representativas, que muitos autores as organizam em gerações ou em décadas de evolução. É com uma mistura destas duas classificações que vamos mostrar, a você, agora, esta evolução histórica.



Vale a pena mencionar que, atualmente, muitos estudos importantes têm sido feitos a respeito dos SO distribuídos, assunto central deste livro. Você estudará detalhadamente sobre eles nas próximas unidades deste material. Por hora podemos adiantar que sistemas distribuídos, basicamente, são representados por vários computadores interligados e, a partir de um computador é possível usar recursos de outros (OLIVEIRA; CARISSIMI; TOSCANI, 2010).

3 DÉCADA DE 1940 OU PRIMEIRA GERAÇÃO

O início da computação, logo no começo da década de 1940, foi marcado pelo desenvolvimento do que poderíamos chamar de máquinas de calcular. Desta época ficaram famosos nomes como Alan Turing, Howard Aiken, John von Neumann, J. Presper Eckert, John Mauchley e Konrad Zuse, que obtiveram sucesso construindo tais máquinas.

A principal causa para o desenvolvimento de máquinas que pudessem, de alguma forma, realizar mais rápido o que era feito de maneira manual, foi a Segunda Guerra Mundial. Em outras palavras, os primeiros computadores, mesmo sendo calculadoras, tinham propósito militar. Tratavam-se de máquinas enormes, que ocupavam espaços imensos (salas inteiras) e formados por milhares de válvulas. Mesmo que a ideia fosse que trabalhassem rápido, seu funcionamento ainda era lento.

Essa época ficou marcada por máquinas famosas como a Colossus (que você pode ver na Figura 6), criada por Alan Turing para decifrar o código das mensagens alemãs gerado pela máquina Enigma; a máquina Mark I, desenvolvida pelo professor Howard Aiken, da Universidade de Harvard; e, talvez, a máquina mais conhecida de todas, o computador ENIAC (*Electronic Numerical Integrator And Calculator*), considerado o primeiro computador digital e eletrônico, desenvolvido pelos engenheiros J. Presper Eckert e John W. Mauchly na Universidade da Pensilvânia.

FIGURA 6 – A MÁQUINA COLOSSUS DE ALAN TURING



FONTE: <<https://www.cbsnews.com/pictures/breaking-the-nazis-enigma-codes-at-bletchley-park>>. Acesso em: 25 out. 2018.

Pode-se imaginar que em máquinas como essas não poderia existir um SO, sendo este um conceito que surgiu somente na década seguinte. Portanto, dispositivos como teclados e monitores, ou algo que permitisse a interface com o usuário também não existiam.

Assim, essa primeira geração foi marcada, principalmente, pela inexistência de linguagens de programação como as que existem atualmente, e máquinas operadas por fios conectados em painéis que controlavam as funções. Então, quando um programa era executado, ele sozinho controlava a máquina durante horas.



Você acabou de ver que, nesta primeira geração, o modo normal de operar o “computador” era plugando conectores no painel. Todavia, você sabia que para fazer isso o programador reservava um horário em uma folha fixada na parede e ficava esperando por horas pela execução do programa e torcendo para que nenhuma válvula queimasse? (TANENBAUM; WOODHULL, 2008).

4 DÉCADA DE 1950 OU SEGUNDA GERAÇÃO

O marco principal que diferenciou os computadores da primeira geração para os computadores da segunda geração foi a evolução da válvula para o transistor. Os transistores foram introduzidos a partir da metade da década de 1950 e possibilitaram o aumento da velocidade e da confiabilidade no processamento dos computadores. Agora sim seria possível fabricar computadores e vendê-los, pois, os computadores continuariam funcionando por tempo suficiente. Não havia mais o problema de válvulas queimadas.

Junto dos transistores veio também a memória magnética, cuja contribuição está no “acesso mais rápido aos dados, maior capacidade de armazenamento e computadores menores” (MACHADO; MAIA, 2017, p. 8).

Os computadores baseados em transistores eram menores que seus antecessores, mas, ainda assim, tinham grande porte, e ficaram conhecidos pelo nome de *mainframe*.

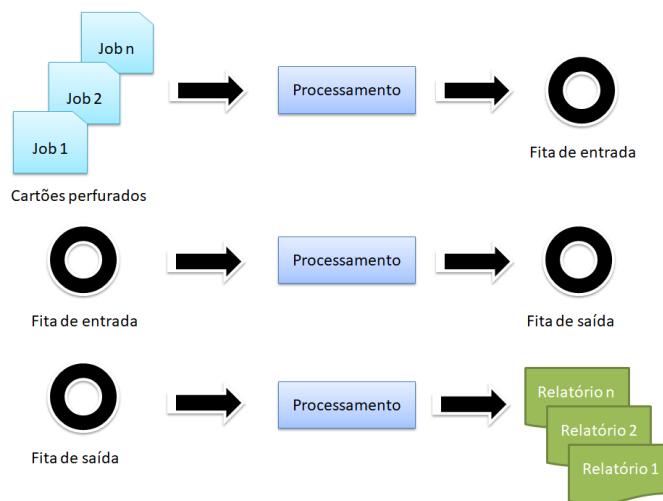


Sobre os *mainframes*, é possível afirmar que: “essas máquinas, [...], eram postas em salas especiais com ar-condicionado e com equipes [...] treinadas para manter-las funcionando. Somente grandes empresas, importantes órgãos do governo ou universidades podiam arcar com seu preço, na casa dos milhões de dólares” (TANENBAUM; WOODHULL, 2008, p. 27)

Em termos de rotinas de trabalho utilizando o computador, essas mudaram no início dos anos 1950 com a introdução dos cartões perfurados (que substituíram os painéis com conectores) para a escrita e leitura de programas (TANENBAUM; WOODHULL, 2008). Ainda assim, os procedimentos que levavam longos períodos de tempo para serem realizados permaneceram os mesmos.

Para resolver o problema do tempo passou-se a submeter não somente um programa (o *job*) de cada vez, mas um conjunto de programas. Surgia então o que passou a se chamar processamento em lote ou *batch*. Veja na Figura 7, como funcionava o processamento em lote:

FIGURA 7 – PROCESSAMENTO EM LOTE OU BATCH



FONTE: Adaptado de Machado e Maia (2017)

Os programas ou *jobs* passaram a ser perfurados em cartões, que, submetidos a uma leitora, eram gravados em uma fita de entrada. A fita, então, era lida pelo computador, que executava um programa de cada vez, gravando o resultado do processamento em uma fita de saída. Ao término de todos os programas, a fita de saída era lida e impressa (MACHADO; MAIA, 2017, p. 8).

O ano de 1953 ficou marcado pelo surgimento do primeiro SO, na ocasião denominado “monitor”. Foi chamado desta forma exatamente por sua simplicidade. O monitor era um programa que permanecia o tempo todo na memória, e tinha como função básica fazer a transição entre programas de forma automática (MACHADO; MAIA, 2011; 2017; OLIVEIRA; CARISSIMI, TOSCANI, 2010).

Esta transição acontecia da seguinte maneira: quando um programa que estava sendo executado terminava, o monitor carregava automaticamente o próximo programa que era então executado. Havia, pelo menos, duas vantagens no monitor: a primeira, a troca entre os programas (transição) feita pelo monitor ao invés de um operador humano era mais rápida, logo vem a segunda vantagem, por causa da transição automática, o tempo que o computador ficava parado entre um programa e outro tornou-se menor (OLIVEIRA; CARISSIMI; TOSCANI, 2010).

Grandes computadores de segunda geração eram usados principalmente para cálculos científicos e de engenharia, como a solução de equações diferenciais parciais que frequentemente ocorrem na física e na engenharia. Eles eram programados principalmente em FORTRAN e em linguagem *assembly*. Sistemas operacionais típicos eram o FMS (o Fortran Monitor System) e o IBSYS, sistema operacional da IBM para o 7094 (TANENBAUM; WOODHULL, 2008, p. 28).

5 DÉCADAS DE 1960 E 1970 OU TERCEIRA GERAÇÃO

Como você percebeu, muito aconteceu após o surgimento dos transistores. Essa evolução no hardware dos computadores permitiu que pelo menos duas coisas muito importantes acontecessem, e que, no mínimo, viabilizaram o avanço tecnológico em que nos encontramos nos dias de hoje: primeiro, a redução nos custos dos computadores, facilitando sua aquisição; e segundo, a difusão do uso do SO (MACHADO; MAIA, 2011; 2017). Nesse sentido, as décadas de 1960 e 1970 foram muito representativas e não há um consenso entre os autores sobre em qual geração ambas estão situadas. Por isso aqui decidimos apresentá-las juntas, agrupando seus conteúdos como terceira geração nesta evolução histórica.

Falando agora, mais especificamente, sobre os avanços relacionados ao SO, a década de 1960 ficou marcada pelo desenvolvimento de funções como o multiprocessamento, a multiprogramação, *time-sharing* e memória virtual. Funções que você já estudou rapidamente no Tópico 1 deste Livro Didático, lembra-se?

Falando sobre o conceito de multiprogramação, seu surgimento só foi possível a partir do primeiro SO, o monitor (que você também já conhece!). Como já comentamos, com o monitor os programas eram executados pelo computador automaticamente um de cada vez, e, sempre que uma operação de E/S era realizada o processador permanecia parado, esperando o término destas operações.



Lembre-se que as operações de E/S são realizadas pelos periféricos (como impressoras, por exemplo) que são dispositivos mais lentos que um processador. Então considere o seguinte: o tempo gasto para se acessar uma leitora de cartões, poderia representar algo em torno de 10.000 instruções não executadas pelo processador, tornando-o ocioso.

A ideia por trás da multiprogramação foi permitir que diversos programas fossem mantidos simultaneamente na memória principal. Desta maneira, seria possível resolver o problema de ociosidade do processador, pois, enquanto um programa estivesse esperando por uma operação de E/S, outro programa poderia ser executado. Então, quando a operação de E/S terminasse, o primeiro programa poderia continuar a ser executado. Percebeu como agora, com a multiprogramação, o tempo e os recursos do computador são melhor aproveitados?

Foi também durante os anos 1960 que o conceito de compartilhamento de tempo, ou *time-sharing* passou a ser desenvolvido. Você deve ter percebido que o conceito de multiprogramação prevê que diversos programas sejam executados pelo processador. Todavia, para que isso ocorra de maneira organizada, esses programas precisam dividir o tempo de acesso a ele. Além disso, em um sistema de *time-sharing* existe mais uma variável a ser considerada: cada usuário possui um computador (também conhecido como terminal) (OLIVEIRA; CARISSIMI; TOSCANI, 2010). Em um ambiente com vários usuários, cada um em seu terminal, cabe ao SO controlar qual programa, de qual usuário está acessando o processador.

Os SO de *time-sharing* vieram a se popularizar na década de 1970. Outra evolução importante desta década foi o multiprocessamento (outra função que você também já conhece) (MACHADO; MAIA, 2011; 2017). O conceito de multiprocessamento difere do conceito de multiprogramação das seguintes formas: (1) existe mais de um processador no computador, o que exige mais controle do SO; (2) com mais de um processador mais de um programa pode ser executado ao mesmo tempo; e (3) o processamento desses programas é mais rápido.

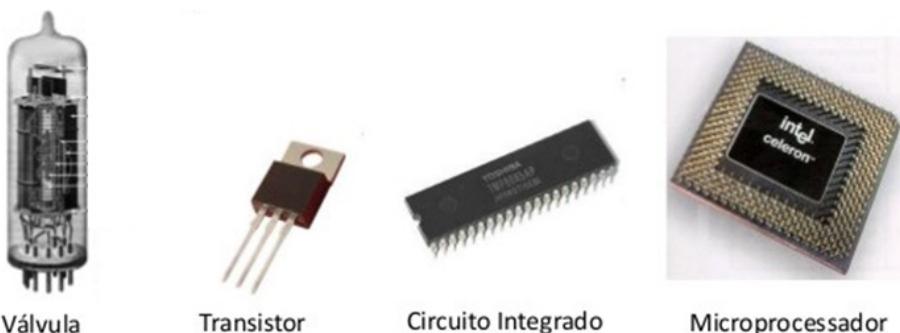
As décadas de 1960 e 1970 merecem destaque ainda pelos lançamentos de computadores que se tornaram famosos. Veja algumas informações importantes:

- Na década de 1960, houve crescimento no desenvolvimento dos minicomputadores, como o PDP-1, lançado em 1961 pela empresa DEC (*Digital Equipment Company*). Um grande sucesso de vendas na época, um PDP-1 custava em torno de US\$120.000.
- Em 1962 o MIT (*Massachusetts Institute of Technology*) desenvolveu o CTSS (*Compatible Time-Sharing System*), SO de tempo compartilhado desenvolvido para um computador IBM 7094. Este SO se tornou a base para outros sistemas de tempo compartilhado como o MULTICS.
- O computador B-5000, da empresa Burroughs, foi lançado em 1963 e contava com um SO chamado MCP (*Master Control Program*), com multiprogramação, memória virtual com segmentação e multiprocessamento assimétrico.
- Em 1971 foi a vez da Intel produzir o Intel 4004, seu primeiro microprocessador, seguido pelo Intel 8080 em 1974. O Intel 8080 foi utilizado no primeiro microcomputador, o Altair.
- A partir do Altair os microcomputadores ganharam o mercado e, o ano de 1976 ficou marcado pela fundação de duas grandes e conhecidas empresas: Apple e Microsoft. Ainda neste mesmo ano, foi produzido o Apple II, por Steve Jobs e Steve Wozniak, tornando-se um sucesso absoluto.
- CP/M (*Control Program Monitor*) da Digital Research era o SO dominante dos microcomputadores na época.

6 DÉCADAS DE 1980 E 1990 OU QUARTA GERAÇÃO

Se entre a primeira e a segunda gerações o grande diferencial esteve na mudança dos computadores baseados em válvulas para os computadores baseados em transistores, a quarta geração já contava com o grande avanço do desenvolvimento dos chips de silício ou circuitos integrados. Era o momento da popularização dos microcomputadores, que, graças à IBM, passaram a ser chamados de computadores pessoais (PC ou *Personal Computer*) baseados em microprocessadores. Veja esta evolução dos componentes dos computadores na Figura 8.

FIGURA 8 – VÁLVULA, TRANSISTOR, CIRCUITO INTEGRADO E MICROPROCESSADOR



FONTE: Adaptado de <<https://pt.slideshare.net/vitorhugod3/aula-2-introduo-aos-conceitos-bsicos>>. Acesso em: 29 out. 2018.

Essa mesma evolução pela qual passou o hardware dos computadores foi acompanhada pelo seu SO. Assim, em 1981 a IBM lançou seu primeiro PC, com processador Intel de 16 bits e um SO chamado DOS (*Disk Operating System*) desenvolvido pela Microsoft. Tanto o DOS como o SO dos computadores Apple chamado Apple DOS eram muito parecidos com o CP/M (sobre o qual você leu agora há pouco), pois todos eram baseados em linhas de comando digitadas pelo usuário, como você pode observar na Figura 9.

FIGURA 9 – INTERFACE DOS SO CP/M, DOS, e APPLE DOS

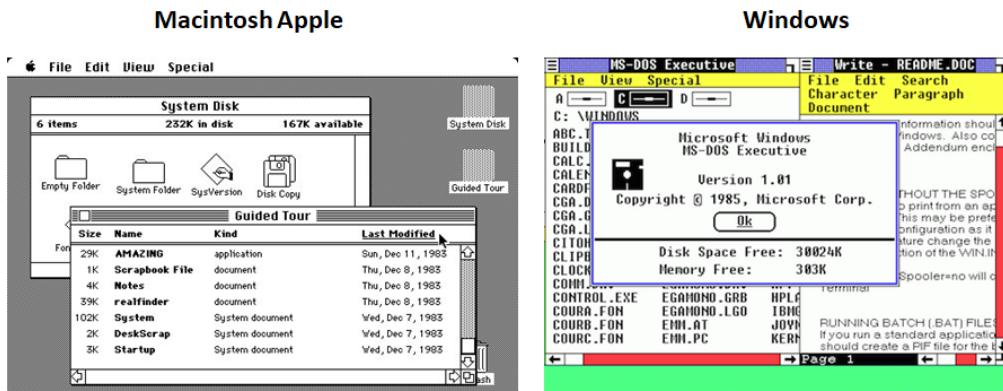


FONTE: Adaptado de <<https://www.tudocelular.com/android/noticias/n73308/A-historia-da-maca-conheca-todas-as-versoes-ja-lancadas-do-macOS.html>>; <http://en.uncyclopedia.co/wiki/MS-DOS>; <http://canacepgd.com/synonym/disk-operating-system.html>>. Acesso em: 29 out. 2018.

Apesar de já ter sido inventada, foi somente com a evolução dos microprocessadores que surgiram os primeiros SO comerciais com interface gráfica, como o Windows da Microsoft e o OS/2 da IBM (MACHADO; MAIA, 2011; 2017). Steve Jobs percebeu que a interface gráfica, com janelas, menus e ícones, tornaria os computadores mais amigáveis para todos, e não somente para programadores. Então, em 1984, a Apple lançou o Macintosh, que tinha "[...] CPU 68000 de 16 bits da Motorola e tinha 64 KB de memória ROM (*Read Only Memory* – memória somente de leitura)" (TANENBAUM; WOODHULL, 2008, p. 32, grifo nosso).

Rapidamente, a Microsoft entrou nesta briga e lançou seu SO com interface gráfica: o Windows (que na verdade não era bem um SO, mas um ambiente gráfico que era executado sobre o DOS). Veja como eram as interfaces do Macintosh e do Windows, na Figura 10, em 1984.

FIGURA 10 – INTERFACES GRÁFICAS DO MACINTOSH E DO WINDOWS



FONTE: Adaptado de <<http://history-computer.com/ModernComputer/Personal/Macintosh.html>; <https://medium.theuxblog.com/a-short-history-of-computer-user-interface-design-29a916e5c2f5>>. Acesso em: 29 out. 2018.

Nesta época, o SO UNIX era considerado um importante concorrente. Porém, seu uso estava mais direcionado para computadores do tipo estações de trabalho e servidores, que exigem alto desempenho.

Também, em meados da década de 1980, merece destaque o desenvolvimento das redes de computadores (e das iniciativas que deram origem à internet), e, consequentemente, dos SO de redes. Neste sentido, ficaram famosos na época os SO como o *Novell Netware* e o *Microsoft LAN Manager* (MACHADO; MAIA, 2011; 2017; TANENBAUM; WOODHULL, 2008).

Os anos de 1990 vieram para consolidar os SO baseados em interface gráfica e muitos “[...] conceitos e implementações só vistos em sistemas considerados de grande porte foram introduzidos na maioria dos sistemas para desktop, como na família Windows da Microsoft e no Unix” (MACHADO; MAIA, 2017, p. 12).

Houve, também, uma evolução considerável não somente no hardware e software dos computadores, mas também na área das telecomunicações. O surgimento da internet no final da década anterior exigiu que os fabricantes de SO se adaptassem ao seu principal protocolo, o TCP/IP (*Transmission Control Protocol/Internet Protocol*). Além disso, com o advento das redes de computadores, uma série de problemas relacionados ao ambiente, como segurança, gerenciamento e desempenho, passaram a estar relacionados ao SO (MACHADO; MAIA, 2011; 2017).

Vale ainda destacar o surgimento do SO Linux, criado por Linus Torvalds em 1991, baseado no SO UNIX. Ao longo dos anos, o Linux vem se tornando uma alternativa muito popular ao Windows não somente no ambiente acadêmico, mas também em grandes empresas.

7 ANOS 2000 ATÉ O PRESENTE

Do início dos anos 2000 para cá consideramos estar vivendo na Era da Informação e da Comunicação, graças a todos os avanços tecnológicos ocorridos. Você deve ter percebido isso: os computadores estão melhores, mais eficientes e com custos bem mais reduzidos do que há vinte ou trinta anos atrás. Além disso, um sistema computacional já não é representado somente pelo PC sobre uma mesa, ele está na mochila, no formato de um notebook, ou no bolso, no formato de um smartphone.

Essa evolução refletiu, também, no funcionamento dos SO. Processadores com arquiteturas de 64 bits permitiram aos SO melhorarem o desempenho na execução das aplicações, especialmente quando precisam manipular grandes volumes de dados. Outro conceito muito difundido é o de processamento distribuído, que você estudará mais à frente neste livro (MACHADO; MAIA, 2011; 2017).

Além disso, por volta dos anos 2010, os SO foram para dentro de outros aparelhos: os chamados dispositivos móveis, como os celulares e tablets, com interfaces ricas e interativas, tornando seu uso inteligente e eficiente. Os celulares já não se limitam mais a fazer e receber chamadas; tornaram-se verdadeiros computadores de bolso.

[...] Com relação aos sistemas operacionais de smartphones, o mercado apresenta diferentes opções entre sistemas proprietários e abertos. Muitos aplicativos têm sido desenvolvidos para estas plataformas. Atualmente os principais sistemas operacionais para smartphones existentes são: Symbian OS, Windows Mobile, Android, Blackberry OS e iOS (MACHADO; MAIA, 2011; 2017, p. 14).

Veja um pouco sobre cada um deles (MACHADO; MAIA, 2011; 2017):

- **Symbian OS:** SO presente nos smartphones Nokia. Trata-se de um SO muito estável, capaz de executar aplicações em tempo real. É muito eficiente e provê boa integração com o computador.
- **Windows Mobile:** SO desenvolvido pela Microsoft com a intenção de proporcionar aos usuários do celular as mesmas características do SO do computador. Dentre suas características está a integração com Microsoft Exchange Server, para sincronia de e-mails, por exemplo, entre o computador e o celular, e também com outros dispositivos, como o Xbox 360.
- **Android:** é um SO aberto desenvolvido por um grupo de 47 empresas, chamado *Open Handset Alliance*, liderado pela Google. O Android suporta diversos padrões de conectividade como *Bluetooth*, redes de dados (como 4G e 4,5 atualmente) e Wi-Fi (*Wireless-Fidelity*).
- **BlackBerry OS:** este é um SO proprietário desenvolvido pela empresa RIM (*Research In Motion*) para sua linha de smartphones BlackBerry. É considerado pioneiro em smartphones para usuários corporativos.
- **iOS:** desenvolvido pela Apple, este SO ficou conhecido por ser utilizado pelo iPhone. Dentre as características que o tornaram muito popular entre os usuários está sua interface, considerada intuitiva e amigável.

RESUMO DO TÓPICO 2

Neste tópico, você aprendeu que:

- Por causa da complexidade em utilizar os primeiros computadores os SO evoluíram para facilitar seu uso.
- A década de 1940 marcou o início da computação com o surgimento dos primeiros computadores, basicamente, máquinas de calcular, de grande porte, de processamento lento, baseadas em válvulas e painéis com conectores.
- A década de 1950 se diferenciou da década anterior pela substituição das válvulas por transistores, aumentando a velocidade e a confiabilidade nos computadores. Na mesma época foram introduzidos a memória magnética e os cartões perfurados para a escrita e leitura de programas.
- Em 1953, surgiu o primeiro SO, chamado monitor, que tinha como função básica fazer a transição entre programas de forma automática.
- As décadas de 1960 e 1970 ficaram marcadas pela redução nos custos dos computadores e pela difusão do uso dos SO. Neste período, foram introduzidos os conceitos de multiprogramação e time-sharing.
- As décadas de 1980 e 1990 marcaram o início da miniaturização dos componentes com a introdução dos circuitos integrados, o consequente barateamento dos custos dos microcomputadores e sua popularização. Nesta mesma época a IBM lançou o primeiro PC e os SO evoluíram de interfaces baseadas em linhas de comando, como o DOS, para interfaces gráficas como o Windows.
- Por fim, os anos 2000 até a época atual registram a era da informação e da comunicação com a drástica melhoria de desempenho do hardware e, consequentemente, de seus SO, bem como a popularização dos dispositivos móveis, como tablets e smartphones.

AUTOATIVIDADE



1 Os mapas mentais são uma técnica utilizada para organizar informações e auxiliar no processo de memorização e compreensão de diversos assuntos. Falamos muitas coisas sobre a evolução dos SO ao longo dos últimos 50, 60 anos. Que tal agora organizar as informações mais importantes sobre cada período para facilitar seu aprendizado? Preencha o mapa mental a seguir com os elementos principais observados em cada período assinalado:



FONTE: Os autores

2 Uma das importantes funções dos SO é: fornecer e manter a interface do usuário, e ela deve ser a mais amigável e intuitiva possível e “simples, confiável e eficiente”. Logo, com a evolução dos SO essa função foi cada vez mais deixada ao controle:

- () Do hardware que também evoluiu juntamente com os SO.
- () Do software, já que apesar da evolução do hardware as manipulações de programas são, ainda, pouco amigáveis.
- () Dos softwares aplicativos, uma vez que a sua evolução permitiu a construção de interfaces de interação mais amigáveis para os usuários.
- () Principalmente dos softwares de sistemas, especialmente o SO, em combinação com a evolução do hardware e dos softwares aplicativos.

3 As primeiras ideias que culminaram com o projeto do computador e a Ciência da Computação, foram apresentadas por mentes com um íntimo desejo de modificar suas realidades, tornando processos demorados e manuais realizados pelo ser humano em processos automatizados. Desde então, muitos outros projetos foram apresentados e implementados até se chegar ao modelo do computador atual. Isso contribuiu para a evolução do hardware, do software e, consequentemente, dos SO. Dessa forma, associe os itens fornecidos com as suas respectivas sentenças corretas:

- I- John Pesper Eckert Jr. () Criador da máquina Colossus, responsável por decifrar as mensagens geradas pela máquina alemã Enigma.
II- Howard Aiken. () Criador do projeto mais conhecido da década de 1940, e considerado o primeiro computador digital e eletrônico.
III- Alan Turing. () Cientista da Universidade de Harvard e criador da máquina Mark I.

A partir das definições fornecidas, a sequência CORRETA para as assertivas é:

- a) () I – III – II.
b) () II – III – I.
c) () III – I – II.
d) () I – II – III.

4 As primeiras ideias que culminaram com o projeto do computador e a Ciência da Computação, foram apresentadas por mentes com um íntimo desejo de modificar suas realidades, tornando processos demorados e manuais realizados pelo Ser Humano em processos automatizados. Desde então, muitos outros projetos foram apresentados e implementados até se chegar ao modelo do computador atual. Isso contribuiu para a evolução do hardware, do software e, consequentemente, dos SO. Dessa forma, assinale a alternativa que indica as assertivas sobre a relação dos primeiros projetos de computadores e os SO:

- I- As primeiras máquinas, década de 1940, já não existiam sem um SO e já contavam com dispositivos como teclados e monitores.
II- As primeiras máquinas, década de 1940, existiam sem um SO e também não contavam com dispositivos como teclados e monitores, sendo que o conceito de SO só surgiu a partir da década de 1950.
III- A década de 1940 também foi marcada pela falta do conceito de linguagens de programação, sendo que as máquinas dessa década eram controladas através da combinação de fios conectados em painéis.
IV- Já a década de 1950 foi marcada pela evolução nas máquinas com a substituição dos transistores pelas válvulas.

A partir das definições fornecidas, estão CORRETAS as assertivas:

- a) () II e III.
- b) () I e IV.
- c) () Somente a III.
- d) () Somente a II.

5 As primeiras ideias que culminaram com o projeto do computador e a Ciência da Computação foram apresentadas por mentes com um íntimo desejo de modificar suas realidades, tornando processos demorados e manuais realizados pelo Ser Humano em processos automatizados. Desde então, muitos outros projetos foram apresentados e implementados até se chegar ao modelo do computador atual. Isso contribuiu para a evolução do hardware, do software e, consequentemente, dos SO. Dessa forma, assinale a alternativa CORRETA sobre a relação dos primeiros projetos de computadores e os SO:

- a) () Na década de 1950 a introdução das válvulas possibilitou o aumento da eficiência no processamento dos computadores
- b) () Aos transistores, década de 1950, juntou-se a memória magnética, com sua contribuição para o aumento da velocidade de acesso aos dados, para a capacidade de armazenamento de dados maior, e a possibilidade da criação de projetos de computadores menores.
- c) () Porém, a geração de 1950 pouco mudou as formas de se utilizar o computador, pois continuou baseada nos chamados cartões perfurados.
- d) () Apesar da geração de 1950 ainda permanecer com a utilização dos cartões perfurados, os processos de interação com os computadores, que levavam longos períodos, passaram a consumir menos tempo.

ORGANIZAÇÃO E ARQUITETURA DE UM SISTEMA OPERACIONAL

1 INTRODUÇÃO

Nos tópicos anteriores você teve a oportunidade de conhecer os conceitos iniciais sobre SO, além de navegar um pouco sobre a história desse tipo especial de software. Agora você terá a oportunidade de conhecer um pouco mais sobre como está organizada a estrutura interna de um SO, chamada também de arquitetura.



A arquitetura, em relação a computadores, software e redes é definida como: "[...] o projeto geral de um sistema computacional e os inter-relacionamentos lógico e físico entre seus componentes. A arquitetura especifica o hardware, software, métodos de acesso e protocolos utilizados pelo sistema".

FONTE: <<https://www.gartner.com/it-glossary/architecture/>>. Acesso em: 5 jul. 2019.

A arquitetura de um SO está diretamente ligada aos elementos de um sistema computacional, apresentados anteriormente, que pode ser considerado o ambiente no qual um SO será executado, além da arquitetura precisar corresponder ao objetivo ou finalidade de um SO. Com isso, podemos concluir que a arquitetura, ou estrutura, de um SO precisa permitir a comunicação entre os elementos computacionais, traduzir seu objetivo e finalidade, tornando o uso do computador mais eficiente e conveniente.

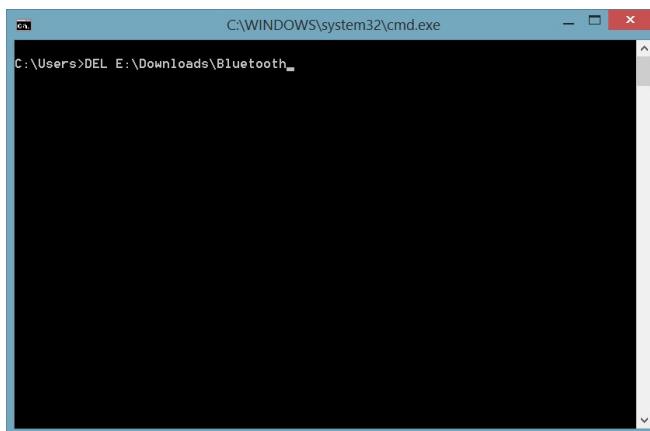
2 SERVIÇOS SUPORTADOS PELO SO

Como você deve lembrar, pois vimos anteriormente, um SO apresenta uma série de funções (serviços) que devem atender tanto às necessidades do usuário quanto permitir aos desenvolvedores programar suas tarefas de maneira mais fácil (SILBERSCHATZ; GALVIN; GAGNE, 2015). Você percebeu novamente a referência à dupla eficiência e conveniência apresentada no Subtópico 2 do Tópico 1.

Assim, ao falarmos da arquitetura e organização de um SO, torna-se importante que você conheça algumas funções mais específicas realizadas por um SO apontadas por (SILBERSCHATZ; GALVIN; GAGNE, 2015):

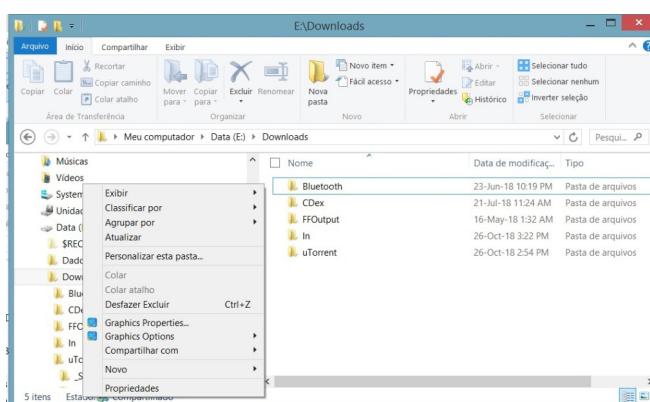
- **Interface de Usuário:** todos os SO precisam de uma interface de usuário (UI - *User Interface*) que permita a execução de comandos iniciados pelos usuários basicamente de três formas: (1) linha de comando (CLI - *Command-line Interface*) que usa comandos de texto através de um teclado, por exemplo; (2) interface *batch*, que são grupos de comandos contidos em arquivos que são executados na forma de lotes; e, finalmente, (3) interface gráfica de usuário (GUI - *Graphical User Interface*) que permite a execução de comandos através de componentes gráficos como janelas, menus e botões que direcionam esses comandos para operações de E/S. As Figuras 11 e 12 mostram um exemplo de uma ação para a exclusão da pasta “Bluetooth” do drive “E” por linha de comando e a mesma ação através de uma interface gráfica usando a opção “Excluir” respectivamente.

FIGURA 11 – EXCLUSÃO POR LINHA DE COMANDO



FONTE: Os autores

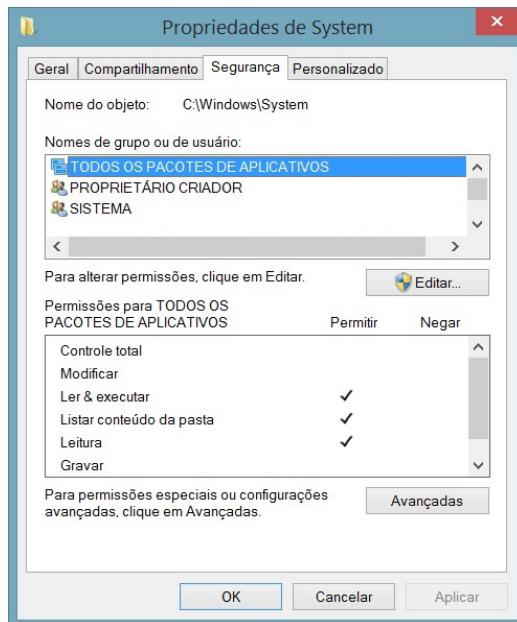
FIGURA 12 – EXCLUSÃO EM INTERFACE GRÁFICA



FONTE: Os autores

- **Execução de programas:** o SO carrega e executa um programa em memória e depois o encerra de maneira normal ou não, e nesse caso deve indicar qual erro ocorreu.
- **Operações de E/S:** uma operação de E/S pode ser realizada a qualquer momento por um programa em execução requerendo um arquivo específico ou algum dispositivo específico como CD, DVD ou mesmo a atualização de uma tela. Por essa razão, pela necessidade de um acesso a estas operações, o SO não pode deixar esse controle em nível de usuário.
- **Manipulação do sistema de arquivos:** todo SO possui um sistema de arquivos, normalmente baseado no conceito de arquivos e diretórios (pastas). A esses são permitidas operações de criação, recuperação, atualização e exclusão, inclusive com a utilização de filtros de pesquisa. O SO pode fornecer um gerenciamento de permissões para esses arquivos e diretórios. Além disso, o SO ainda pode fornecer várias opções de sistemas de arquivos como: Windows (FAT, FAT16, FAT32, NTFS), Linux (Ext2, Ext3, ReiserFS, Linux Swap - FAT16, FAT32, NTFS), MacOS (HFS, MFS). A Figura 13 mostra o controle de permissão para uma pasta da família de SO Windows.

FIGURA 13 – CONTROLE DE PERMISSÕES NO WINDOWS



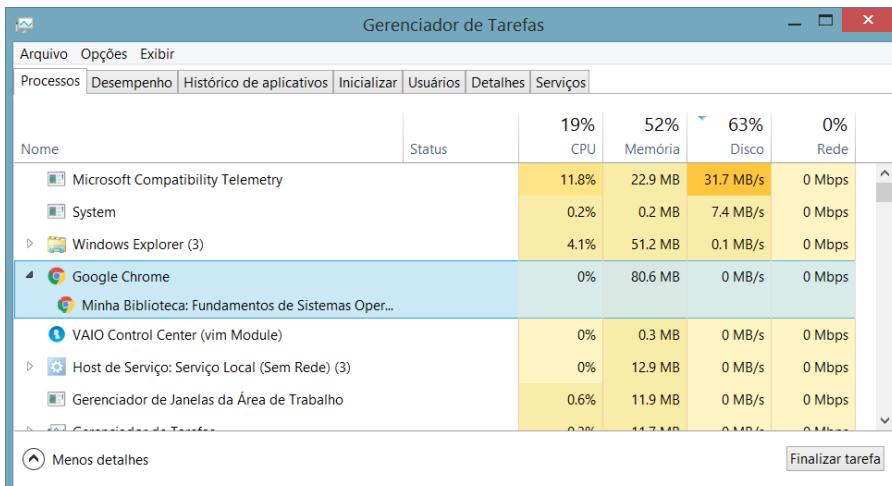
FONTE: Os autores

- **Comunicações:** os processos criados pelos programas em execução sobre um SO, normalmente precisam trocar informações entre si, isto é, precisam se comunicar. Tais processos podem estar rodando em um ambiente no qual as máquinas compartilham uma memória em comum ou mesmo em um ambiente de rede através de sistemas computacionais diferentes. Seja através de mensagens entre sistemas diferentes, ou de memórias compartilhadas, os processos em execução em um SO precisam se comunicar.

- Detecção de erros:** o SO deve tratar os mais variados tipos de erros: (1) aqueles gerados por uma falha de hardware (CPU ou memória), (2) aqueles gerados por operações de dispositivos de E/S, e também (3) aqueles gerados pelos programas dos usuários. Esses erros podem ser causados por eventos como falta de energia, discos rígidos, ou pen drives defeituosos. E, na ocorrência de um evento, o SO deve tomar a decisão do que fazer com o erro gerado: (1) interromper o sistema; (2) encerrar o processo causador do erro; (3) retornar uma mensagem com um código de erro, ou até mesmo uma combinação de ações, entre outras decisões.

Se algum dos processos em execução no SO Windows, mostrado na Figura 14, apresentar algum erro, o sistema deve ser capaz de identificar qual a medida mais adequada para o encerramento desse processo, se ele está associado a outro processo, se o mesmo representa um processo de usuário ou se representa um processo do próprio SO.

FIGURA 14 – PROCESSOS EM EXECUÇÃO NO SO WINDOWS



FONTE: Os autores



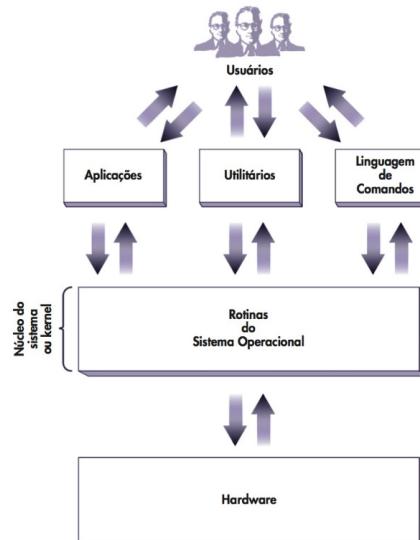
O seu celular também possui um sistema de arquivos, já que contém um SO Android ou IOS. Na opção Android você pode usar o gerenciador de arquivos Files Go da Google.

Os serviços mencionados anteriormente estão mais relacionados ao objetivo de auxiliar o usuário em suas operações e programas executados no SO. Porém, existem serviços que, além desse objetivo, têm como foco garantir o funcionamento eficiente do próprio sistema. São eles (SILBERSCHATZ; GALVIN; GAGNE, 2015):

- **Alocação de recursos:** este serviço é essencial para um SO na medida que é responsável pela distribuição dos processos dos usuários e do próprio sistema, de recursos essenciais (CPU, memória principal e armazenamento de arquivos, por exemplo) e de recursos menos prioritários, como dispositivos de E/S. Este serviço toma uma proporção bem mais complexa se pensarmos em sistemas com múltiplos usuários e múltiplos recursos compartilhados (como você estudará mais adiante no livro).
- **Contabilização:** permite a contagem da quantidade e do tipo de recursos acumulada pelos usuários e seus processos permitindo um controle de liberação de recursos quando necessário, ou apenas para estatística dos recursos que estão sendo utilizados para uma melhor configuração e reconfiguração dos recursos do sistema.
- **Proteção e segurança:** sistemas com multiusuários ou em rede estão sujeitos a uma ampla concorrência por recursos e compartilhamento de dados. Dessa forma, um processo ou operação de um determinado usuário não pode interferir no processo ou operação de outros usuários ou mesmo dos processos do SO, isto é, o acesso a qualquer recurso do sistema deve ser controlado (protegido), além da necessidade de se evitar a entrada de invasores aos processos do SO. É importante entender que essa proteção e segurança são estendidos a dispositivos externos de E/S como, por exemplo, adaptadores de rede.

Como você deve ter percebido esses serviços estão em conformidade com o Quadro 2. Tais serviços são distribuídos pelas camadas da estrutura de um SO e suas interfaces na Figura 15.

FIGURA 15 – ESTRUTURA E INTERFACE DO SO



FONTE: Machado e Maia (2017, p. 47)

Dessa forma, os serviços de um SO são acessados através das rotinas implementadas por esse sistema. Então, lembre-se que ao grupo rotinas dá-se o nome de núcleo do sistema, isto é, *kernel*, é ele quem responde às solicitações de serviços mostrados anteriormente neste livro. Existem três formas do usuário se comunicar com o *kernel*: (1) usando aplicações de softwares, (2) utilitários (forma mais amigável) ou (3) através de linhas de comando (forma mais avançada). As principais funções do *kernel* estão apresentadas no Quadro 2.

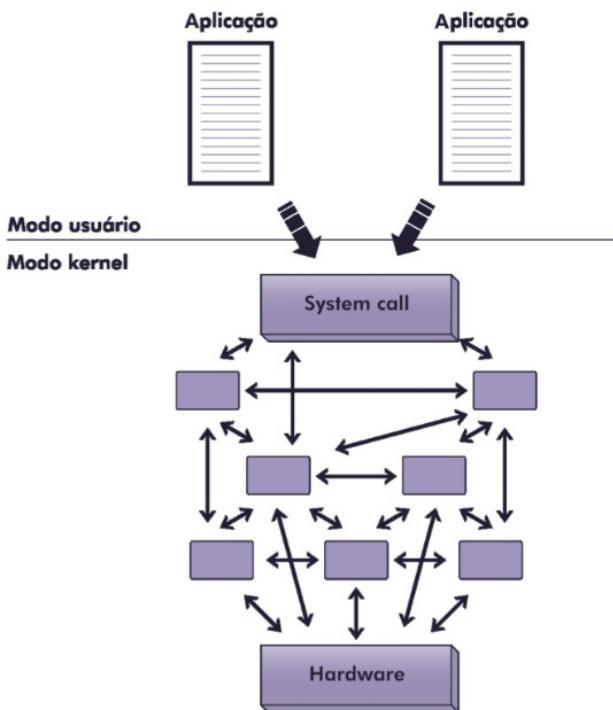
3 ARQUITETURAS SUPORTADAS PELO SO

Agora que você já conhece as principais funções de um SO, distribuídas através de seus principais serviços, aprenderá na sequência que tais serviços são construídos sobre um dado modelo de arquitetura que podem ser: arquitetura monolítica, arquitetura em camadas, máquinas virtuais e arquiteturas *microkernel*.

3.1 ARQUITETURA MONOLÍTICA

Organização mais direta para a concepção de um SO, porém, mais desorganizada e ineficiente conhecida como “a grande bagunça”, e que alguns autores chamam também de Estrutura Simples, na qual o SO é construído com um conjunto de rotinas que podem ser chamadasumas pelas outras sem qualquer critério. E, sempre que uma rotina nova é necessária, ela apenas é criada e adicionada ao conjunto já existente, formando um imenso programa único e executável.

FIGURA 16 – ARQUITETURA MONOLÍTICA



FONTE: Machado e Maia (2017, p. 54)

Você pode perceber que nessa arquitetura não há uma separação entre as rotinas do SO e nem entre os elementos computacionais “usuário” e “hardware” do ambiente no qual o SO está executando. Essa estrutura foi aplicada para os primeiros sistemas operacionais Unix e para a construção do SO DOS. Atualmente, essa estrutura ainda apresenta indícios de sua presença em versões dos sistemas Unix, Linux e Windows.

3.2 ARQUITETURA EM CAMADAS

Essa arquitetura pode ser considerada uma versão mais aprimorada da estratégia presente na arquitetura monolítica já representada na Figura 16, tendo como objetivo organizar a estrutura do SO em níveis de camadas, uma construída sobre a outra. A primeira abordagem dessa arquitetura foi a concepção do sistema THE (*Technische Hogeschool Eindhoven*), construído por Edsger Dijkstra no final da década de 1960, na Holanda, e que utilizava seis camadas hierárquicas.

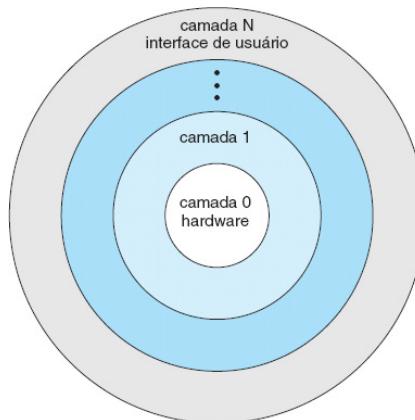


Edsger Dijkstra, nome completo: Edsger Wyber Dijkstra (nascido em 11 de maio de 1930, Rotterdam, Holanda e falecido em 6 de agosto de 2002, em Nuenen, também na Holanda), foi um cientista da computação. Recebeu seu Ph.D. da Universidade de Amsterdam enquanto trabalhava no Amsterdam's Mathematical Center (1952-62).

FONTE: <<https://www.britannica.com/biography/Edsger-Dijkstra>>. Acesso em: 5 jul. 2019.

A separação do modelo monolítico em camadas permite que o SO tenha um controle muito maior sobre um computador (não esqueça que os serviços de um SO estão distribuídos utilizando o conceito de camadas do próprio sistema computacional apresentado, anteriormente, uma vez que, se tratando de SO modulares, as camadas permitem a especialização (ocultação) dos dados manipulados. Isto é, rotinas de alto nível (softwares aplicativos dos usuários) não se misturam com rotinas de baixo nível (hardware) o que não acontece no modelo monolítico. Veja a concepção da ideia de camadas mostrada na Figura 17.

FIGURA 17 – ARQUITETURA EM CAMADAS



FONTE: Silberschatz, Galvin e Gagne (2015, s. p.)

No modelo em camadas cada uma oferece funcionalidades utilizadas apenas pelas camadas superiores. Isto é, a camada mais inferior (camada 0 – hardware) oferece funções para a camada 1, e a camada de mais alto nível de hierarquia (camada N), fornece funcionalidades para utilização pelo usuário. Porém, implementar esse modelo não é uma tarefa trivial, na verdade trata-se de um desafio bem mais complexo do que construir um modelo monolítico, já que uma abordagem em camadas envolve, obrigatoriamente, a especificação adequada de todas as funções pertinentes que serão executadas por cada uma das camadas.

Além disso, uma separação em camadas provoca a desvantagem de uma preocupação adicional com o desempenho, já que cada camada precisa se comunicar (conversar) com a sua camada superior, ou inferior. Nos dias atuais, a maioria dos SO, maioria das versões do Unix e do Windows da Microsoft, utiliza um modelo baseado em duas camadas, nas quais existem os modos de acesso usuário (não privilegiado) e *kernel* (privilegiado) para a separação das funcionalidades dos serviços do SO.

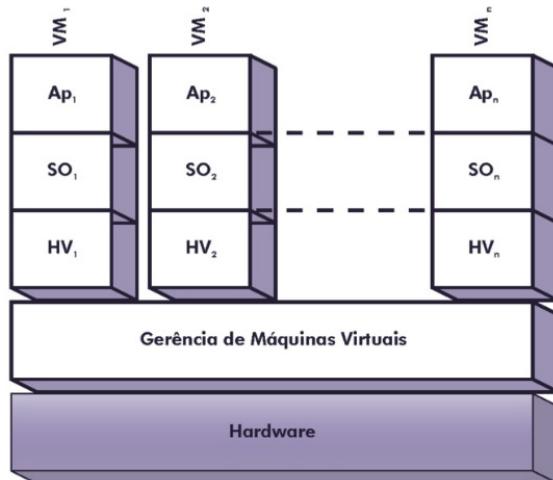
3.3 MÁQUINAS VIRTUAIS

Nesta arquitetura cada máquina virtual (VM, do inglês: *virtual machine*) é idêntica ao hardware verdadeiro; cada uma pode executar qualquer SO que seja executado diretamente no hardware básico. Diferentes VM podem executar (e frequentemente executam) diferentes SO. Diversas implementações de VM são vendidas comercialmente, como, por exemplo: *VMWare*, *Virtual PC*, *VirtualBox*.

Outra área onde as máquinas virtuais são usadas, mas de uma maneira um tanto diferente, é na execução de programas Java. Quando a Sun Microsystems inventou a linguagem de programação Java, inventou também uma máquina virtual (isto é, uma arquitetura de computador) chamada JVM (Java Virtual Machine – máquina virtual Java). O compilador Java produz código para a JVM, o qual então é normalmente executado por um interpretador JVM, em software (TANENBAUM; WOODHULL, 2008, p. 63).

Como você já sabe, um sistema computacional é formado por níveis, em que a camada de nível mais baixo é o hardware. Acima desta camada encontramos o SO que oferece suporte para as aplicações, como visto na Figura 15. O modelo de VM cria um nível intermediário entre o hardware e o sistema operacional, denominado gerência de máquinas virtuais, como você pode observar na Figura 18. Este nível cria diversas VM independentes, na qual cada uma oferece uma cópia virtual do hardware, incluindo os modos de acesso, interrupções, dispositivos de E/S, entre outros.

FIGURA 18 – MÁQUINAS VIRTUAIS



FONTE: Machado e Maia (2017, p. 56)

Como cada VM é independente das demais, é possível que cada uma delas tenha seu próprio sistema operacional e que seus usuários executem suas aplicações como se todo o computador estivesse dedicado a cada um deles.

Este modelo cria o isolamento total entre cada VM, oferecendo grande segurança para cada uma. Se, por exemplo, uma VM executar uma aplicação que comprometa o funcionamento do seu SO, as demais não sofrerão qualquer problema.

A seguir, elencamos algumas vantagens na utilização de VM (MACHADO; MAIA, 2017):

- **Portabilidade de código:** normalmente um programa executável precisa de hardware específico para que seja executado. Isso o torna dependente daquela plataforma em específico. Quando se faz uso de uma VM, é feita a geração de código de programação intermediário. Desta forma, é possível que o programa seja portado para qualquer ambiente, sem que haja a necessidade de reescrevê-lo.
- **Consolidação de servidores:** representa uma melhor utilização dos recursos de hardware, como CPU, memória principal e dispositivos de E/S pelo SO. A consolidação permite que várias aplicações que são processadas em servidores diferentes, venham a ser executadas em um único sistema computacional utilizando uma VM para cada aplicação.

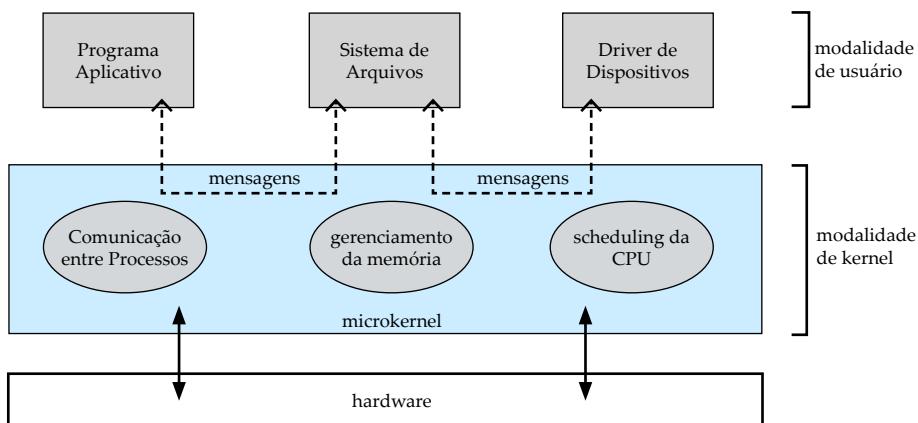
- **Aumento da disponibilidade:** uma VM pode ser copiada facilmente de um disco rígido para outro, inclusive em uma máquina diferente, e restaurada para que o seu ambiente possa ser novamente disponibilizado.
- **Facilidade de escalabilidade e balanceamento de carga:** ao apresentar sinais de baixo desempenho, uma VM pode ser transferida para um ambiente com maior capacidade de processamento.
- **Facilidade no desenvolvimento de software:** ambientes de testes para softwares podem ser criados independentes do ambiente de produção no qual esses softwares estão sendo executados, impedindo, com isso, que o ambiente de produção seja comprometido com testes.

3.4 ARQUITETURAS MICROKERNEL

Com o aumento da amplitude dos softwares de usuários e sua complexidade a partir da evolução da computação, o *kernel* teve que responder a esse aumento tornando-se mais difícil de gerenciar. Partindo desse novo cenário, a abordagem de *microkernel* reestrutura o SO retirando componentes não essenciais do *kernel* e adicionando-os como programas de nível de sistema e de usuário diminuindo assim o tamanho do *kernel*.

Nesta estrutura, os *microkernel* gerenciam um grupo mínimo dos processos e da memória, juntamente com recursos de comunicação, tendo como sua função base estabelecer a comunicação entre os programas de usuário e os diversos serviços executados no seu espaço. A Figura 19 apresenta um modelo para essa arquitetura.

FIGURA 19 – ARQUITETURA MICROKERNEL



FONTE: Silberschatz, Galvin e Gagne (2015, s. p.)

O modelo baseado em *microkernel* propicia uma extensão mais natural de um SO, uma vez que os serviços novos podem ser adicionados ao espaço do usuário, e, consequentemente, não requerem a modificação do *kernel*. E quando uma modificação for necessária, elas tendem a ser menores já que o *microkernel* é um *kernel* menor. Veja agora alguns exemplos de SO que utilizam a arquitetura de *microkernel*: Tru64 UNIX, Mac OS X (ou *Darwin*) e o legado Windows NT 4.0.

RESUMO DO TÓPICO 3

Neste tópico, você aprendeu que:

- A arquitetura de um SO está diretamente ligada aos elementos de um sistema computacional, e precisa corresponder ao objetivo do SO.
- Os SO devem exercer uma série de funções, também chamadas serviços, que devem atender tanto às necessidades dos usuários quanto facilitar o trabalho de programadores.
- A interface com o usuário é um destes serviços e permite que o usuário execute programas a partir de três formas: por intermédio de linhas de comando, por interfaces batch, e interfaces gráficas, sendo estas últimas as mais amigáveis.
- São quatro as arquiteturas suportadas pelos SO: monolítica, em camadas, máquinas virtuais e *microkernel*.
- Em uma arquitetura monolítica, também conhecida pela expressão “a grande bagunça”, não há uma separação entre as rotinas do SO e nem entre os elementos computacionais “usuário” e “hardware” do ambiente no qual o SO está executando.
- Na arquitetura em camadas a estrutura do SO está organizada em níveis de camadas, uma construída sobre a outra. Nesta arquitetura a comunicação acontece entre as camadas: a camada mais inferior (camada 0 – hardware) oferece funções para a camada 1, e a camada de mais alto nível de hierarquia (camada N), fornece funcionalidades para utilização pelo usuário.
- A arquitetura de máquinas virtuais cria um nível intermediário entre o hardware e o sistema operacional, basicamente, oferecendo uma cópia virtual do hardware, incluindo os modos de acesso, interrupções, dispositivos de E/S, entre outros.
- A arquitetura *microkernel* foi responsável por reestruturar o SO diminuindo o tamanho do seu kernel, a partir da retirada de componentes não essenciais e adicionando-os como programas de nível de sistema.



- 1 Neste tópico, você viu que a arquitetura de um SO está diretamente ligada aos elementos de um SO. Desta forma, a arquitetura, ou estrutura, de um SO precisa permitir a comunicação entre os elementos computacionais, traduzir seu objetivo e finalidade, tornando o uso do computador mais eficiente e conveniente. Isto é possível a partir da execução de certos serviços específicos por parte do SO. Na sequência são apresentadas algumas definições destes serviços. Classifique em V para as sentenças verdadeiras e F para as falsas.
- () A interface com o usuário (*UI – User Interface*) permite a execução de comandos iniciados pelos usuários, mas nem todos os SO precisam de uma interface de usuário.
 - () Linha de Comando (*CLI – Command-Line Interface*) que usa comandos de texto através de um teclado; interface batch, que são grupos de comandos contidos em arquivos que são executados na forma de lotes; e interface gráfica de usuário (*GUI – Graphical User Interface*) são as três formas de interface com o usuário oferecidas pelo SO.
 - () Os processos criados pelos programas em execução sobre um SO, normalmente, precisam trocar informações entre si, isto é, precisam se comunicar, seja através de mensagens entre sistemas diferentes, ou de memórias compartilhadas.
 - () O serviço de detecção de erros permite ao SO tratar os mais variados tipos de erros, causados por eventos diversos. E, na ocorrência de um evento, o SO deve tomar a decisão do que fazer com o erro gerado.
- 2 Você aprendeu que o SO é um software de sistema responsável por realizar uma série de funções importantes para o funcionamento de um sistema computacional. As funções executadas pelo SO somente são possíveis porque são implementadas sobre um dado modelo de arquitetura, podendo ser: arquitetura monolítica, arquitetura em camadas, máquinas virtuais e arquiteturas *microkernel*. Sobre as diversas arquiteturas de um SO, assinale a alternativa INCORRETA:
- a) () A arquitetura em camadas pode ser considerada uma versão mais aprimorada da estratégia presente na arquitetura monolítica, uma vez que no modelo em camadas cada uma oferece funcionalidades utilizadas apenas pelas camadas superiores.
 - b) () Uma máquina virtual é idêntica ao hardware verdadeiro; cada uma pode executar qualquer SO que seja executado diretamente no hardware básico. Diferentes VM podem executar diferentes SO.
 - c) () A arquitetura monolítica é a organização mais direta para a concepção de um SO, e a mais organizada e eficiente. Alguns autores a chamam de “estrutura simples”, porque o SO é construído com um conjunto de rotinas que podem ser chamadasumas pelas outras.
 - d) () Em uma arquitetura de *microkernel* o SO é reestruturado, retirando-se componentes não essenciais do *kernel* e adicionando-os como programas de nível de sistema e de usuário diminuindo assim o tamanho do *kernel*.

3 O conceito de arquitetura perpassa pelo “[...] projeto geral de um sistema computacional e os inter-relacionamentos lógico e físico entre seus componentes”. Logo, o projeto eficiente de um SO está associado à escolha de uma arquitetura adequada às execuções de seus processos e serviços. Sobre a importância das arquiteturas para o projeto de um SO, assinale a alternativa CORRETA:

- a) () A arquitetura possibilita a troca de informações entre os elementos computacionais, precisa estar relacionada ao objetivo e finalidade do SO, porém não tem relação com o quesito uso do computador mais amigável no tocante às necessidades do usuário final.
- b) () A arquitetura prioriza a troca de informações entre os elementos computacionais, especialmente em nível da camada de interação com o hardware, deixando os aspectos relacionados à camada de interação com os sistemas aplicativos em um plano secundário.
- c) () A arquitetura possibilita a troca de informações entre os elementos computacionais, precisa estar relacionada ao objetivo e finalidade do SO, e torna o uso do computador mais amigável produzindo o efeito esperado para o usuário.
- d) () A arquitetura prioriza a troca de informações entre os elementos computacionais, especialmente em nível da camada de interação com os softwares aplicativos, deixando os aspectos relacionados à camada de interação com o hardware em um plano secundário.

4 Um SO apresenta uma série de funções (serviços) que devem atender tanto às necessidades do usuário quanto permitir aos desenvolvedores programar suas tarefas de maneira mais fácil. Dessa forma, no contexto dessas funções, existem serviços com características bem específicas (operações de usuário e programas) executados pelos SO. Tendo isso em mente, assinale a alternativa CORRETA sobre esses serviços.

- a) () Uma operação de E/S é realizada apenas em momentos específicos definidos pelo SO sem que um dado programa em execução requeira algum tipo de serviço ou acesso ao SO. Por essa razão, pela necessidade de um acesso a estas operações, o SO não pode deixar esse controle em nível de usuário.
- b) () Pode-se considerar quase uma regra geral que os processos criados pelos programas em execução sobre um SO necessitam trocar informações entre si. Esses processos, rodando em um ambiente de memória compartilhada ou em um ambiente interconectado por sistemas computacionais diferentes, podem necessitar trocar mensagens nesses ambientes.
- c) () Um SO carrega e executa um programa em memória apenas se houver garantias de que ele encerra de maneira normal, sem a necessidade de indicar se algum erro ocorreu.
- d) () Um SO trata alguns tipos de erros como falhas de hardware (CPU ou memória), erros gerados por operações de dispositivos de E/S, porém aqueles gerados pelos programas dos usuários são tratados exclusivamente por esses programas para que não haja sobrecarregas nas atividades do SO.

- 5 Um SO apresenta uma série de funções (serviços) que devem atender tanto às necessidades do usuário quanto permitir aos desenvolvedores programar suas tarefas de maneira mais fácil. Dessa forma, além dessas funções, existem serviços com características bem mais específicas, associados à garantia de bom funcionamento do próprio sistema. Tendo isso em mente, assinale a alternativa CORRETA sobre esses serviços.
- a) () Os SO sempre oferecem algum tipo interface de usuário, baseada em gráficos ou em apenas texto, com a função de executar comandos submetidos pelos usuários através de linha de comando, interface *batch*, interface gráfica de usuário, ou mesmo uma combinação dessas formas.
 - b) () Um SO, normalmente, possui seu conceito de sistemas de arquivos baseados em arquivos e diretórios (pastas) físicos, aos quais são submetidas operações de criação, consulta, modificação e exclusão, incluindo a utilização de filtros de pesquisa sobre algumas dessas operações.
 - c) () Uma outra função exercida pelo SO, porém não relacionada ao funcionamento eficiente do sistema, permite a contagem (quantidade e tipo) de recursos iniciados pelos usuários e seus processos. Essa função permite o gerenciamento dos recursos no tocante a sua liberação quando necessária, ou apenas para fins estatísticos, buscando uma melhor configuração e reconfiguração dos recursos do sistema.
 - d) () A alocação de recursos é, sem dúvida, um dos serviços mais essenciais presentes em um SO. O gerenciamento desse serviço permite a distribuição correta de recursos (processador, uso de memória principal e manipulação de arquivos, dispositivos de E/S, entre outros) aos processos dos usuários e do próprio sistema. E se idealizarmos ambientes mais complexos com múltiplos usuários e múltiplos recursos compartilhados, esse gerenciamento toma uma proporção ainda mais complexa e bem mais delicada.

TIPOS DE SISTEMAS OPERACIONAIS

1 INTRODUÇÃO

Já evoluímos bastante em nossos estudos sobre SO. Veja só, a esta altura você já sabe que:

- Um SO é um software de sistema que atua como um intermediário entre o hardware e o usuário em um sistema computacional (o que é?).
- Tem como objetivo fornecer ao usuário um ambiente conveniente e eficiente para a execução de seus programas (para que serve?).
- Dentre suas várias funções, o SO deve controlar o hardware e os periféricos do computador e também garantir que as operações do sistema acontecerão de maneira correta (como funciona?).

Além disso, tivemos a oportunidade de mostrar a você uma evolução histórica dos SO, e foi possível perceber que sua evolução sempre caminhou junto com o avanço dos hardwares dos computadores.

No tópico anterior, você mergulhou um pouco mais fundo no estudo sobre o SO e conheceu sobre sua arquitetura, e pôde perceber que os SO variam em sua organização por muitas linhas diferentes. Por isso mesmo o projeto de um SO é uma tarefa bastante complexa e ampla.

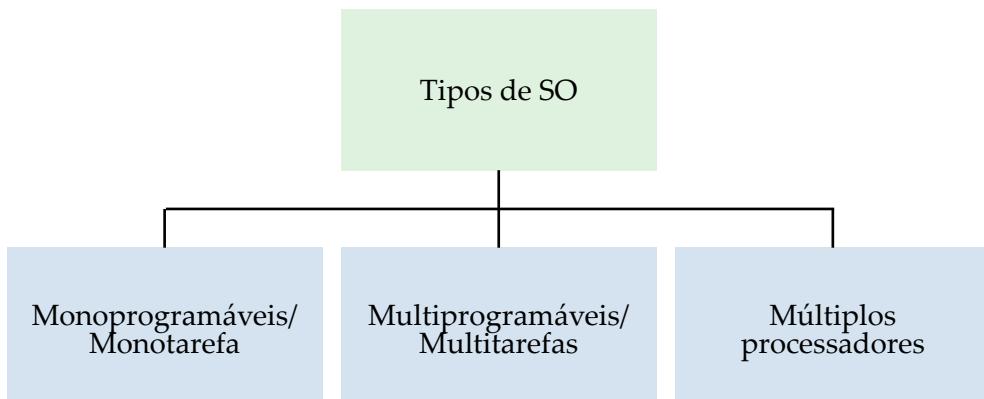
O projeto de um novo SO está diretamente relacionado a sua finalidade, ou seja, é muito importante que se saiba para que será feito o SO (objetivos bem definidos). Com objetivos bem definidos torna-se mais simples estabelecer de que tipo será o SO, assunto que abordaremos a partir de agora.

Com a evolução do hardware dos computadores e depois da criação do primeiro SO (lembra-se do CP/M?), novos conceitos foram introduzidos ao estudo e desenvolvimento dos SO, decorrentes das necessidades relacionadas de como os usuários poderiam executar suas aplicações.

Os tipos de sistemas operacionais e sua evolução estão relacionados diretamente com a evolução do hardware e das aplicações por ele suportadas. Muitos termos inicialmente introduzidos para definir conceitos e técnicas foram substituídos por outros, na tentativa de refletir uma nova maneira de interação ou processamento. [...] Inicialmente, os termos programa ou *job* eram os mais utilizados, depois surgiu o conceito de processo e subprocesso e, posteriormente, o conceito de thread (MACHADO; MAIA, 2017, p. 15, grifo nosso).

Desta forma, os SO foram evoluindo e classificados em tipos, à medida que se distinguiam em características, e sua própria evolução se encarregava de evidenciar suas vantagens e desvantagens. Por exemplo, é possível classificar os SO pelo número de usuários (monousuários e multiusuários) e pelo seu nível de complexidade (monotarefa e multitarefa). Há ainda os SO que trabalham com múltiplos processadores, aqueles que suportam aplicações em tempo real, e mais recentemente os SO especialmente desenvolvidos para os dispositivos móveis. A Figura 20 mostra os principais tipos de SO já organizados.

FIGURA 20 – TIPOS DE SO



FONTE: Adaptado de Machado e Maia (2011; 2017)



Esta é uma representação simples dos tipos de SO. Pode haver subdivisões dentro delas, e, sempre que necessário, estas subdivisões serão tratadas no livro.

De maneira geral, os SO desenvolvidos para dispositivos móveis são construídos para serem utilizados por um único usuário; já os SO para computadores do tipo desktop (os PCs) tanto aceitam um único usuário, quanto um pequeno grupo deles; por fim, os SO residentes em servidores de pequenas e grandes empresas aceitam um número maior de usuários, que pode variar de alguns poucos até dezenas ou mesmo centenas (TURBAN; RAINER JR.; POTTER, 2003).

2 MONOTAREFA

Este tipo de SO está relacionado aos primeiros computadores digitais, aqueles mesmos que você estudou no Subtópico 3 do Tópico 2, sendo, na época, utilizados por um único usuário.

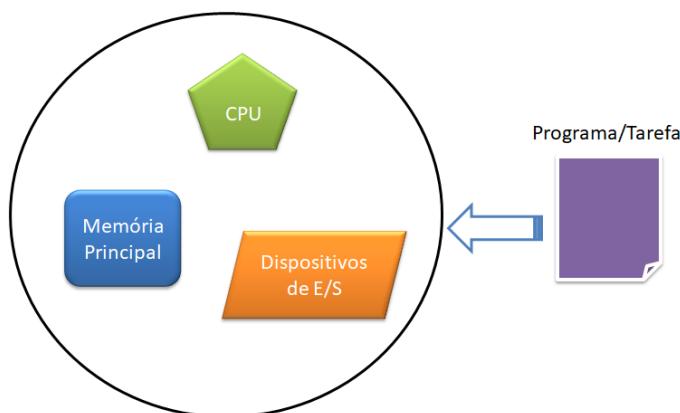
Por definição, um SO monotarefa, ou monoprograma, ou monoprogramável, como ainda pode ser conhecido, é um tipo de sistema no qual o processador, a memória e os periféricos ficam dedicados à execução de um programa de cada vez. Isto significa dizer que se o processador está executando um programa, um novo programa deve esperar pelo término deste primeiro (MACHADO; MAIA, 2017).



Como o sistema precisa aguardar o término de um programa para executar o próximo, o processador fica ocioso. Consequentemente, a memória acaba por ser subutilizada e os periféricos nem sempre são utilizados.

Veja como está organizado este tipo de SO na Figura 21:

FIGURA 21 – SO MONOTAREFA OU MONOPROGRAMÁVEL



FONTE: Adaptado de Machado e Maia (2017)

“Comparados a outros sistemas, os sistemas monoprogramáveis ou monotarefa são de simples implementação, não existindo muita preocupação com problemas decorrentes do compartilhamento de recursos, como memória, processador e dispositivos de E/S” (MACHADO; MAIA, 2017, p. 16).

3 MULTITAREFA

Os chamados SO multitarefa, ou multiprogramáveis, como também são conhecidos, representam a evolução dos SO monoprogramáveis que você acabou de estudar. Esta capacidade de multiprogramação, aliás, passou a ser uma das características mais importantes dos SO. A razão para isso é simples:

com SO multiprogramáveis passou a ser possível compartilhar os recursos de processamento, memória e periféricos entre vários usuários e aplicações (MACHADO; MAIA, 2017; SILBERSCHATZ; GALVIN; GAGNE, 2015).

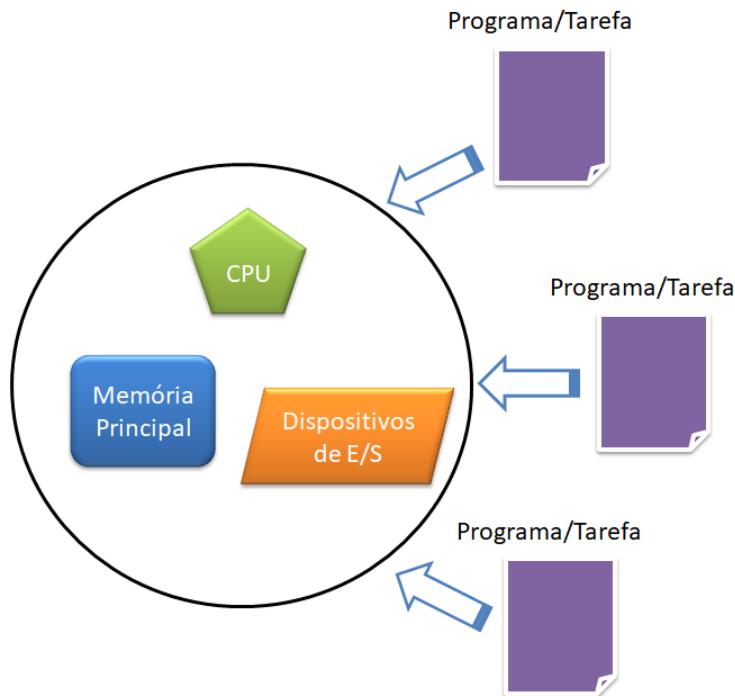
O funcionamento de um SO multiprogramável se dá da seguinte maneira: para que sejam processados, os programas são encaminhados para o processador, mas antes disso precisam ser carregados na memória. A questão é que a maioria dos programas não precisa de toda a memória, mas apenas de uma pequena parte dela. Assim, para que a memória não seja subutilizada como acontecia nos SO monoprogramáveis, basicamente, o SO aloca o espaço da memória com vários programas, até que todos sejam executados pelo processador.

Este funcionamento pode ser mais detalhado da seguinte forma:

[...] enquanto um programa espera por uma operação de leitura ou gravação em disco, outros programas podem estar sendo processados neste mesmo intervalo de tempo. Nesse caso, podemos observar o compartilhamento da memória e do processador. O sistema operacional se preocupa em gerenciar o acesso concorrente aos seus diversos recursos, como memória, processador e periféricos, de forma ordenada e protegida, entre os diversos programas (MACHADO; MAIA, 2017, p. 16).

Veja a organização de um SO multiprogramável na Figura 22.

FIGURA 22 – SO MULTITAREFA OU MULTIPROGRAMÁVEL



FONTE: Adaptado de Machado e Maia (2017)



Em outras palavras, diferentemente dos SO monoprogramáveis, agora a CPU, a memória ou os dispositivos de E/S não ficam mais ocupados o tempo todo por um único programa: enquanto um programa está usando a impressora, por exemplo, o processador já pode estar sendo usado por outro programa diferente.

Os SO multiprogramáveis possuem uma implementação mais complexa se comparados aos monoprogramáveis, porém como você pôde perceber, seu uso mostra-se mais vantajoso, seja pela redução do tempo de execução das aplicações, seja pelo compartilhamento de recursos, o que acarreta em redução de custos.

Uma outra característica importante sobre os SO multiprogramáveis é que eles podem ser classificados em função do número de usuários que o utilizam em monousuário e multiusuário, como demonstra o Quadro 3.

QUADRO 3 – TIPO DE SO E USUÁRIOS

	Um usuário	Dois ou mais usuários
Monoprogramável/ Monotarefa	Monousuário	N/A
Multiprogramável/ Multitarefa	Monousuário	Multiusuário

FONTE: Adaptado de Machado e Maia (2017, p. 16)

Perceba que um SO monoprogramável, necessariamente, permite a existência de um único usuário. Já no caso de um sistema multiprogramável, o mesmo é monousuário quando uma única pessoa acessa o sistema computacional e utiliza vários programas ao mesmo tempo. Esta situação é muito comum nos SO de PCs ou de dispositivos móveis como o celular, por exemplo. Neste caso, somente você está interagindo com seu celular e utilizando vários aplicativos ao mesmo tempo. O sistema multiprogramável é multiusuário quando há vários usuários conectados simultaneamente, como em uma empresa, por exemplo.

Além desta classificação por quantidade de usuários, os SO multiprogramáveis podem ser classificados em sistemas *batch*, sistemas de tempo compartilhado ou sistemas de tempo real, dependendo da forma como cada um gerencia suas aplicações. É importante ressaltar que um SO é capaz de suportar uma ou mais destas classificações.

- **Sistemas *batch*:** originados nos anos 1960 (como você já acompanhou anteriormente em nosso material). Atualmente, já não existem sistemas exclusivamente dedicados a este processamento. Quando implementados em um SO, sistemas em *batch* são caracterizados por serem executados sem exigir a interação do usuário com a aplicação que acaba sendo executada automaticamente por alguma memória secundária. São exemplos de aplicações em *batch* cálculos numéricos, compilações, ordenações e backups.
- **Tempo compartilhado:** também já falamos sobre eles anteriormente! Este tipo de sistema, também chamado *time-sharing* caracteriza-se pela divisão de tempo do processador (fatias de tempo) para a execução de diversos programas, criando, para cada usuário, um ambiente próprio, passando a impressão de que todo o sistema está sendo usado exclusivamente por ele. Por esta razão ficaram conhecidos como sistemas on-line.
- **Tempo real:** estes sistemas são implementados de forma semelhante aos sistemas de tempo compartilhado, sendo que o que os diferencia é exatamente o tempo necessário para a execução dos programas. Nos sistemas de tempo real não existe o conceito de fatia de tempo, logo, um programa utiliza o tempo que for necessário para sua execução até que apareça outro programa com maior prioridade. Situações como estas podem comprometer as aplicações em execução, o que torna sistemas de tempo real críticos, pois devem estar dentro de limites rígidos de tempo de execução. Exemplos de sistemas de tempo real incluem sistemas de monitoramento de refinarias de petróleo, controle de tráfego aéreo, de usinas termoelétricas e nucleares.

4 MÚLTIPLOS PROCESSADORES

Antes de abordar os SO multiprocessados, é importante lembrar que a maioria dos sistemas computacionais utiliza um único processador. É o caso, por exemplo, do seu aparelho de celular ou de seu notebook. A questão é que existe uma grande variedade de SO monoprocessados, exatamente porque os sistemas computacionais variam desde um aparelho celular até um *mainframe* de uma grande empresa (SILBERSCHATZ; GALVIN; GAGNE, 2008).



No SO de um processador único, ou monoprocessado, existe uma CPU principal capaz de executar as instruções do computador, incluindo instruções dos softwares aplicativos do usuário (SILBERSCHATZ; GALVIN; GAGNE, 2008).

Os sistemas de múltiplos processadores, ou multiprocessados, ou ainda, sistemas paralelos (como também são conhecidos), tem como principal característica o fato de possuírem dois ou mais processadores (CPUs), comunicando-se e trabalhando perfeitamente em conjunto.

Vale lembrar que a principal diferença entre os sistemas multiprocessados e a multiprogramação (que você acabou de estudar) é que no caso dos primeiros ocorre o processamento simultâneo entre os vários processadores, ao passo que na multiprogramação existe uma única CPU realizando o processamento paralelo dos programas (TURBAN; RAINER JR.; POTTER, 2003).

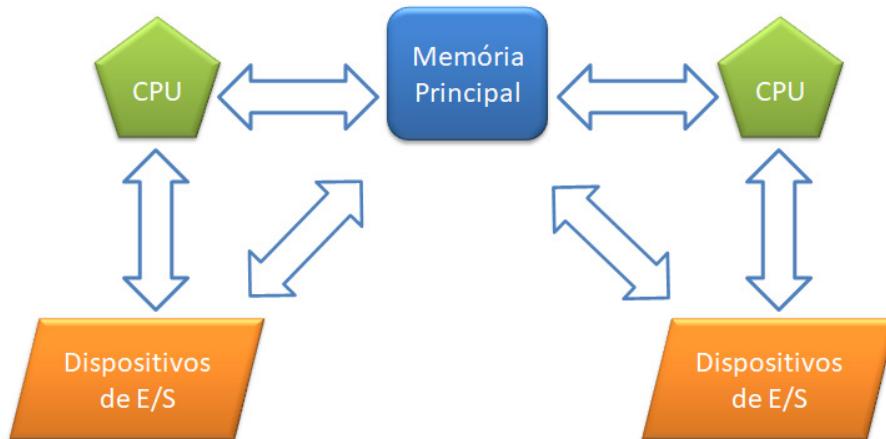
Os sistemas multiprocessados têm ganhado importância cada vez maior em função de alguns aspectos como: custos elevados no desenvolvimento de processadores mais rápidos; desenvolvimento de aplicações que exigem grande poder computacional, como é o caso, por exemplo, de sistemas de previsão do tempo. Com vários processadores, tais questões podem ser resolvidas (MACHADO; MAIA, 2011; 2017).

Além das questões anteriormente levantadas, os sistemas multiprocessados apresentam ainda outras vantagens:

- **Mais velocidade (*throughput*):** com o aumento do número de processadores, espera-se realizar mais trabalho em menos tempo. Isso porque é possível que vários programas sejam executados ao mesmo tempo, ou que um mesmo programa seja dividido em partes e executado ao mesmo tempo por vários processadores. Contudo, é importante salientar que o aumento de velocidade não é proporcional ao aumento do número de processadores. Por exemplo, a existência de quatro processadores em um sistema computacional não representa um aumento de velocidade em quatro vezes. Isto acontece porque é gerado um custo adicional (chamado de *overhead*) que passa a existir para que todos os recursos que concorrem pelo processamento, funcionem corretamente.
- **Economia de escala:** sistemas de múltiplos processadores podem custar menos que vários sistemas computacionais monoprocessados equivalentes, já que no caso dos sistemas multiprocessados há o compartilhamento de dispositivos de armazenamento de massa e fonte de alimentação, por exemplo.
- **Confiabilidade e disponibilidade:** como as funções realizadas são distribuídas igualmente entre os vários processadores, no caso de falha de um deles, o sistema computacional continuará trabalhando normalmente. O que acontecerá é que as funções que estavam sendo realizadas pelo processador que falhou serão divididas entre os demais. Tudo acontece de forma transparente para o usuário.
- **Escalabilidade:** em sistemas multiprocessados é possível aumentar o poder computacional apenas acrescentando novos processadores, o que não é possível em um sistema monoprocessado. Neste caso, quando se deseja melhorar o desempenho é necessária a substituição de todo o sistema computacional por um com maior poder computacional.
- **Balanceamento de carga:** “é a possibilidade de distribuir o processamento entre os diversos processadores da configuração a partir da carga de trabalho de cada processador, melhorando, assim, o desempenho do sistema como um todo” (MACHADO; MAIA, 2017, p. 19).
- A partir do entendimento de suas características e principais vantagens, vamos agora falar sobre os tipos de sistemas multiprocessados. Existem dois: os sistemas fortemente acoplados e os sistemas fracamente acoplados.

Um sistema fortemente acoplado (do inglês: *tightly coupled*) é caracterizado por existirem vários processadores com seus periféricos sendo gerenciados por um único SO. Esses vários processadores compartilham uma única memória. Veja como é a organização deste tipo de sistema na Figura 23:

FIGURA 23 – SISTEMAS FORTEMENTE ACOPLADOS



FONTE: Adaptado de Machado e Maia (2017)



Por causa de suas características os sistemas fortemente acoplados são identificados como sinônimos para sistemas multiprocessados.

Existem dois tipos de sistemas fortemente acoplados: os SMP (*Symmetric Multiprocessors* ou multiprocessamento simétrico) e os sistemas de multiprocessamento assimétrico:

- **SMP:** exatamente por serem sistemas simétricos, todos os processadores são iguais e executam todas as tarefas em um SO, ou seja, os processadores possuem as mesmas funções, não havendo relação de mestre-seguidor (antigamente chamado de mestre-escravo). Além disso, sua principal característica está no fato de os vários processadores acessarem a memória em parcelas uniformes de tempo.
- **Assimétricos:** a assimetria destes sistemas estabelece que cada processador possui uma tarefa específica, existindo um processador que controla o sistema, denominado "mestre". Sempre que um outro processador precisa executar uma tarefa, o "mestre" deverá ser consultado. Esta característica criou a definição de sistemas "mestre-seguidor".

Cabe ainda mencionar que o multiprocessamento pode alterar a maneira como os processadores acessarão a memória, criando, ainda, dois outros tipos de sistemas: os UMA (*Uniform Memory Access* ou acesso uniforme à memória) e os NUMA (*Non-uniform Memory Access* ou acesso não uniforme à memória):

- **UMA:** o acesso uniforme prevê o mesmo tempo de acesso a qualquer memória do sistema a partir de qualquer processador.
- **NUMA:** são assim chamados porque “apresentam diversos conjuntos reunindo processadores e memória principal, sendo que cada conjunto é conectado aos outros através de uma rede de interconexão” (MACHADO; MAIA, 2017, p. 20) o que interfere no tempo de acesso à memória é que pode variar de acordo com a localização física.

Terminado o contexto sobre os sistemas multiprocessados fortemente acoplados, é hora de compreender como funcionam os sistemas fracamente acoplados (*loosely coupled*). Esses são formados por vários sistemas computacionais independentes, cada um com seu SO, gerenciando seus próprios recursos (como CPU, memória e periféricos) (MACHADO; MAIA, 2017). A comunicação entre os vários sistemas computacionais se dá por intermédio de linhas de comunicação, como você pode observar na Figura 24:

FIGURA 24 – SISTEMAS FRACAMENTE ACOPLADOS



FONTE: Adaptado de Machado e Maia (2017)



Por causa de sua organização em vários sistemas computacionais interligados, os sistemas fracamente acoplados também são chamados multicamputadores. Vale ainda lembrar que cada sistema computacional individual pode ser multiprocessado.

A grande diferença entre os dois tipos de sistemas é que em sistemas fortemente acoplados existe apenas uma memória principal sendo compartilhada por todos os processadores, enquanto nos fracamente acoplados cada sistema tem sua própria memória individual. Além disso, a taxa de transferência entre processadores e memória em sistemas fortemente acoplados é muito maior que nos fracamente acoplados (MACHADO; MAIA, 2017, p. 20).

Os modelos de sistemas multiprocessados que existiam até a metade da década de 1980 eram, de modo geral, baseados em estruturas de mestre-seguidor na qual, os terminais dos usuários (chamados “terminais burros” porque não possuíam poder de processamento) estavam conectados ao computador mestre centralizado.

O avanço tecnológico posterior a este período, como você já estudou, contribuiu para um novo modelo chamado redes de computadores. Uma rede de computadores é formada por dois ou mais sistemas computacionais (que dentro da rede são chamados de *hosts*), cada um com capacidade de processamento, interconectados de modo que possam trocar informações e serviços entre si. Em um modelo de rede de computadores a informação deixa de ser centralizada (MACHADO; MAIA, 2017).

Dependendo do grau de integração dos *hosts* na rede, podem existir os SO de rede ou os sistemas distribuídos (MACHADO; MAIA, 2017, p. 21).

- **Sistemas operacionais de rede:** neste caso os *hosts* da rede podem compartilhar recursos uns com os outros. É o que acontece, por exemplo, em um departamento de uma empresa que compartilha uma impressora para seus colaboradores.
- **Sistemas distribuídos:** de maneira geral, os sistemas distribuídos dividem aplicações em partes permitindo que cada parte seja executada em diferentes *hosts* da rede, de modo transparente para o usuário.



Um exemplo bastante comum de sistemas distribuídos são os *clusters*. “Em um *cluster* existem dois ou mais servidores ligados, normalmente, por algum tipo de conexão de alto desempenho. O usuário não conhece os nomes dos membros do *cluster* e não sabe quantos são. Quando ele precisa de algum serviço, basta solicitar ao *cluster* para obtê-lo” (MACHADO; MAIA, 2017, p. 22, grifo nosso).

RESUMO DO TÓPICO 4

Neste tópico, você aprendeu que:

- Os tipos de sistemas operacionais e suas evoluções estão relacionados diretamente com a evolução do hardware e das aplicações por ele suportadas.
- Os SO podem ser classificados pelo número de usuários (monousuários e multiusuários) e pelo nível de complexidade (monotarefa e multitarefa), e também pela quantidade de processadores.
- Os SO monoprogramáveis ou monotarefas são caracterizados por executar um programa de cada vez. Um SO monotarefa necessariamente é monousuário.
- Os SO multiprogramáveis ou multitarefas permitem compartilhar os recursos de processamento, memória e periféricos entre vários usuários e aplicações. Neste caso, o SO busca gerenciar o acesso concorrente aos seus diversos recursos, como memória, processador e periféricos, de forma ordenada e protegida, entre os diversos programas.
- Os SO multitarefa podem ser monousuário, ou seja, suportar somente um usuário, ou multiusuário, suportando mais de um usuário simultaneamente.
- Exemplos de SO multiprogramáveis envolvem os sistemas *batch*, de tempo compartilhado e de tempo real.
- Os sistemas multiprocessados são caracterizados pela existência de mais de um processador e têm ganhado espaço devido aos custos elevados no desenvolvimento de processadores mais rápidos e também por causa do desenvolvimento de aplicações que exigem grande poder computacional.
- Dentre as vantagens dos sistemas multiprocessados estão o aumento de velocidade, a economia de escala, confiabilidade, disponibilidade, escalabilidade e balanceamento de carga.
- Existem dois tipos de sistemas multiprocessados: os fortemente acoplados (que podem ser simétricos e assimétricos) e os fracamente acoplados.
- A diferença entre um sistema fortemente acoplado e um sistema fracamente acoplado é que no primeiro existe apenas uma memória principal sendo compartilhada por todos os processadores, enquanto no segundo cada sistema tem sua própria memória individual.



1 Neste tópico, você compreendeu que “os tipos de sistemas operacionais e sua evolução estão relacionados diretamente com a evolução do hardware e das aplicações por ele suportadas” (MACHADO; MAIA, 2017, p. 15). As assertivas a seguir trazem definições de diferentes tipos de SO vistos neste tópico:

- I- Passou a ser possível compartilhar os recursos de processamento, memória e periféricos entre vários usuários e aplicações.
- II- Tem como principal característica o fato de possuírem dois ou mais processadores comunicando-se e trabalhando perfeitamente em conjunto.
- III- É um tipo de sistema no qual o processador, a memória e os periféricos ficam dedicados à execução de um programa de cada vez.

As definições apresentadas referem-se a quais tipos de SO respectivamente?

- a) () Monotarefa; Multitarefa e Multiprocessado.
- b) () Monotarefa; Monousário e Multitarefa.
- c) () Multitarefa; Multiprocessado e Monotarefa.
- d) () Multiusuário; Multiprocessado e Multitarefa.

2 Um dos tipos de SO que você estudou são os sistemas com múltiplos processadores ou multiprocessados. A principal característica deste tipo de SO é a existência de dois ou mais processadores trabalhando conjuntamente. Por esta razão, estes sistemas são também chamados de sistemas paralelos. Os sistemas multiprocessados apresentam uma série de características que os tornam mais atraentes em relação aos sistemas com um único processador. Sobre os sistemas multiprocessados assinale a alternativa INCORRETA:

- a) () Com o aumento do número de processadores, aumenta-se a velocidade ou throughput, pois espera-se realizar mais trabalho em menos tempo. Isso porque é possível que vários programas sejam executados ao mesmo tempo, ou que um mesmo programa seja dividido em partes e executado ao mesmo tempo por vários processadores.
- b) () Em sistemas multiprocessados é possível aumentar o poder computacional apenas acrescentando novos processadores, o que os torna altamente escaláveis.
- c) () Basicamente existem dois tipos de sistemas multiprocessados: os sistemas fortemente acoplados e os sistemas fracamente acoplados.
- d) () Um sistema fracamente acoplado é caracterizado por existirem vários processadores com seus periféricos sendo gerenciados por um único SO. Esses vários processadores compartilham uma única memória.

3 A construção de um SO envolve vários aspectos relevantes, por exemplo, a sua utilização pelos usuários, o tipo de serviço que deve ser provido por ele e a complexidade destes serviços, ou o tipo de ambiente no qual eles irão rodar, como, por exemplo, os atuais ambientes para dispositivos móveis. Logo, esses aspectos levam a um agrupamento dos SO baseado em uma classificação. Considerando essa classificação assinale a alternativa CORRETA:

- a) () No tocante à natureza da forma de controle exigido pelos serviços dos usuários, uma possível classificação para os SO reside em ele ser ou não multiusuário.
- b) () No tocante ao número de usuários acessando um serviço, ou um recurso, uma possível classificação para os SO reside em ele ser ou não multitarefa.
- c) () No tocante à natureza da forma de controle exigido pelos serviços dos usuários, uma possível classificação para os SO reside em ele ser ou não multitarefa.
- d) () No tocante à natureza da forma de controle exigido pelos serviços dos usuários, uma possível classificação para os SO reside em ele ser ou não monousuário.

4 Um dos grandes desafios para o projeto de um SO reside em garantir que os seus serviços tenham acesso seguro e confiável aos recursos gerenciados por esse SO, sendo que estes recursos estão relacionados ao acesso e compartilhamento do processador, memória e dispositivos de E/S, entre outros. Com isso, a forma como esses recursos são compartilhados e a forma como são acessados caracteriza algumas tipificações para os SO. Considerando essas tipificações assinale a alternativa CORRETA:

- a) () O tipo de sistema no qual o conjunto processador, memória e dispositivos E/S está disponível aos vários programas de usuários um por vez, como em uma espécie de fila, caracteriza um sistema multitarefa. Neste ambiente o SO enfileira os acessos a estes recursos e, como existem vários programas enfileirados, isto caracteriza um ambiente multitarefa.
- b) () O tipo de sistema no qual o conjunto processador, memória e dispositivos E/S está disponível aos vários programas de usuários um por vez, como em uma espécie de fila, caracteriza um sistema multitarefa. Neste ambiente o SO enfileira os acessos a estes recursos e, como existem vários programas enfileirados, isto caracteriza um ambiente de multiprogramação.
- c) () O tipo de sistema no qual o conjunto processador, memória e dispositivos E/S está disponível aos vários programas de usuários um por vez, como em uma espécie de fila, caracteriza um sistema multitarefa. Neste ambiente o SO enfileira os acessos a estes recursos e, como existem vários programas enfileirados, isto caracteriza um ambiente monotarefa.
- d) () O tipo de sistema no qual o conjunto processador, memória e dispositivos E/S está disponível aos vários programas de usuários um por vez, como em uma espécie de fila, caracteriza um sistema multitarefa. Neste ambiente o SO enfileira os acessos a estes recursos e, como existem vários programas enfileirados, isto caracteriza um ambiente de multiprocessamento.

- 5 Um dos grandes desafios para o projeto de um SO reside em garantir que os seus serviços tenham acesso seguro e confiável aos recursos gerenciados por esse SO, sendo que esses recursos estão relacionados ao acesso e compartilhamento do processador, memória e dispositivos de E/S, entre outros. E esse acesso e compartilhamento ganham uma maior importância e complexidade quando o SO tem que lidar com ambientes de multiprocessamento. Considerando essa característica assinale a alternativa CORRETA:
- a) () Ambientes multiprocessados *tightly coupled systems* constituem uma tipificação de sistemas na qual sua natureza de memórias individuais, para cada processador, configura sistemas independentes, porém coesos, comunicando-se para a execução dos seus serviços.
 - b) () Ambientes multiprocessados *loosely coupled systems* constituem uma tipificação de sistemas na qual sua natureza de memórias individuais, para cada processador, configura sistemas dependentes, e sem coesão, mas que se comunicam para a execução dos seus serviços.
 - c) () Ambientes multiprocessados *tightly coupled systems* constituem uma tipificação de sistemas na qual sua natureza de memória individual compartilhada, configura sistemas independentes, porém coesos, comunicando-se para a execução dos seus serviços.
 - d) () Ambientes multiprocessados *loosely coupled systems* constituem uma tipificação de sistemas na qual sua natureza de memórias individuais, para cada processador, configura sistemas independentes, porém coesos, comunicando-se para a execução dos seus serviços.

GERENCIAMENTO DE PROCESSOS

1 INTRODUÇÃO

Ao longo desta unidade tratamos de vários contextos relacionados aos SO. Você encontrou as respostas às três perguntas fundamentais que formulamos: “o que é?”, “como funciona?” e “para que serve?” um SO. A partir das respostas, seguimos além e ainda falamos um pouco da história dos SO, afinal, foi esta história que permitiu sua evolução, uma evolução caracterizada em sua arquitetura e nos tipos de SO existentes, assunto que você também estudou.

Como uma coisa puxa a outra, e nada que esteja relacionado a um SO acontece de maneira isolada dentro do sistema computacional, para falarmos do assunto principal deste tópico, o gerenciamento de processos, você perceberá que para aprender coisas novas estará relembrando de outras que já estudou neste livro.

Antes de mais nada, é importante então entendermos que o gerenciamento de processos é uma das principais funções de um SO, conforme você deve lembrar do Subtópico 3 do Tópico 1. Essa função somente se tornou possível a partir da evolução pela qual os SO passaram ao longo das últimas décadas.

Como você deve recordar, inicialmente os computadores executavam somente um programa de cada vez, ou seja, eram monoprogramáveis (ou monotarefas). Atualmente, os sistemas computacionais modernos executam vários programas simultaneamente, ou seja, são multiprogramáveis (ou multitarefa). Isso significa que enquanto o computador executa um programa, o usuário também pode estar lendo um arquivo de um pen drive e gerando uma saída de um documento na impressora. Em outras palavras, um computador pode fazer várias coisas diferentes ao mesmo tempo.

A capacidade de multiprogramação tornou os SO mais complexos e esta complexidade está diretamente relacionada ao aumento da capacidade do SO de funcionar de acordo com as necessidades de seus usuários. Lembra-se? Um SO deve ser eficiente e conveniente! Para isso o SO passou a exercer um controle mais rígido sobre a execução dos diversos programas do usuário. Deste controle surgiu o conceito de processo.

De maneira simplificada, processo pode ser entendido como um programa em execução e é considerada a unidade de trabalho em um SO multiprogramado. Basicamente, tudo dentro de um SO depende do entendimento desse conceito!

A gerência de um ambiente multiprogramável é função exclusiva do sistema operacional que deve controlar a execução dos diversos programas e o uso concorrente do processador e demais recursos. Para isso, um programa ao ser executado deve estar sempre associado a um processo. O conceito de processo é a base para a implementação de um sistema multiprogramável (MACHADO; MAIA, 2011; 2017, p. 61).

Por esta razão já havíamos comentado neste livro que o gerenciamento de processos é uma das funções essenciais exercidas pelo SO. É a partir dele que o computador permite aos programas compartilhar dados, alocar o uso de recursos (como uma impressora, por exemplo) e sincronizar suas execuções, além de uma série de outras atividades como:

- Executar o escalonamento de processos e *threads* nas CPUs.
- Criar e excluir processos de usuário e do sistema.
- Suspender e retomar processos.
- Fornecer mecanismos de sincronização de processos.
- Fornecer mecanismos de comunicação entre processos.

Vale ainda lembrar que o gerenciamento de processos existe não somente nos sistemas multiprogramáveis (quando os processos compartilham o uso do processador), mas também em sistemas multiprocessados. Neste caso o gerenciamento de processos se encarrega não somente da concorrência entre os processos pelo uso do processador, mas também se encarrega de organizar a execução simultânea dos processos em diferentes processadores (MACHADO; MAIA, 2011; 2017).

2 PROCESSOS

Para que se possa entender como ocorre o gerenciamento de processos dentro de um sistema computacional é importante, antes de mais nada, entender o conceito de processo (o que é?). A partir deste entendimento ficará mais fácil compreender como os processos são gerenciados pelo SO.

Até este ponto do material vimos utilizando com frequência o termo programa, como em expressões do tipo “os programas executados pelo usuário”. É muito comum, mesmo na área computacional, que as pessoas utilizem os termos programa e processos como sinônimos. Isto não representa exatamente um pecado capital, mas é importante saber que existe uma diferença, e mais importante ainda compreender esta diferença para que você não se engane mais daqui para frente.

Então, a primeira coisa que você precisa entender é que um programa, por si só, não é um processo. Um programa nada mais é que um arquivo, armazenado em disco no seu computador, que possui algumas instruções. Enquanto ele não for invocado pelo processador, ele continuará a ser um programa. A partir do momento que este programa é executado pela CPU, então ele passa a ser reconhecido como um processo (SILBERSCHATZ; GALVIN; GAGNE, 2015).



Um processo é entendido como um programa em execução!

Veja o exemplo a seguir comparando uma situação do seu cotidiano com a definição processo:

Considere um profissional de computação com dotes culinários que está assando um bolo de aniversário [...]. Ele tem uma receita de bolo e uma cozinha bem equipada, com os ingredientes necessários: farinha, ovos, açúcar, essência de baunilha, etc. Nessa analogia, a receita é o programa (isto é, um algoritmo expresso em alguma notação conveniente), o profissional de computação é o processador (CPU) e os ingredientes do bolo são os dados de entrada. O processo é a atividade que consiste em nosso confeiteiro ler a receita, buscar os ingredientes e assar o bolo (TANENBAUM; WOODHULL, 2008, p. 69).

Fácil, não é? Veja agora alguns exemplos de processos reais executados pelo seu sistema computacional (SILBERSCHATZ; GALVIN; GAGNE, 2015):

- Um programa de processamento de texto, sendo executado por um usuário individual em um PC, é um processo.
- Uma tarefa do sistema, como o envio de saída para uma impressora, é um processo.
- Um usuário utilizando um navegador *web* e um pacote de e-mail são processos.
- E mesmo que o usuário possa executar apenas um programa de cada vez, como em um dispositivo embutido que não suporte multitarefa, é um processo.

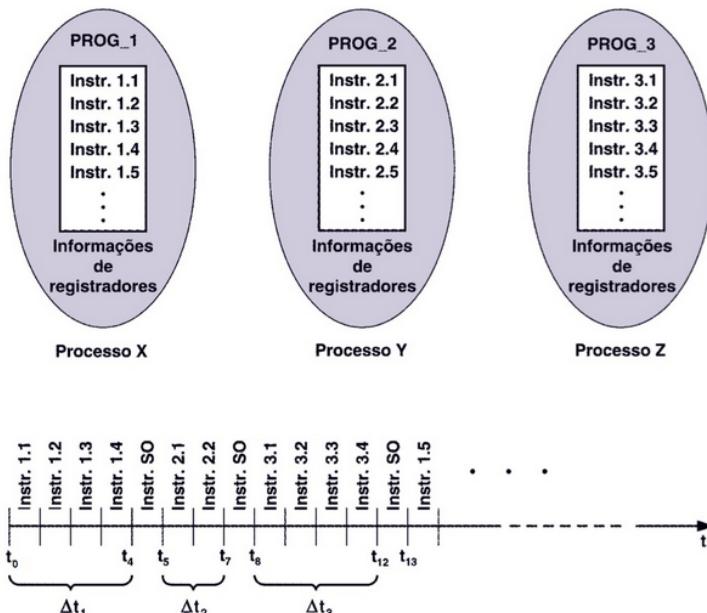
Perceba, então, que processos representam a matéria-prima de trabalho de um SO, independentemente do tipo (se monoprogramável ou multiprogramável, monousuário ou multiusuário). Existem pelo menos dois tipos de processos: os processos do próprio SO e os processos do usuário. E todos eles podem ser executados concorrentemente.



Para que processos sejam executados concorrentemente sem problemas, informações sobre os programas são guardadas. Essas informações servem para que, quando um processo for interrompido para a execução de um segundo processo, o primeiro possa voltar posteriormente a ser executado (após o término do segundo), e não faltem informações para esta continuação. Essa alternância entre processos é a chave dos SO multiprogramáveis.

Esta situação de concorrência prevista pela multiprogramação pode ser visualizada na Figura 25:

FIGURA 25 – CONCORRÊNCIA ENTRE PROCESSOS



FONTE: Machado e Maia (2017, p. 62)

A Figura 25 traz três programas concorrentes (PROG_1, PROG_2 e PROG_3) cada qual associado ao seu processo (Processo X, Processo Y e Processo Z, respectivamente). A linha de tempo demonstra o tempo que cada programa tem suas instruções executadas pelo processador. Então, perceba que no intervalo de tempo Δt_1 , as instruções do PROG_1 são executadas. Na sequência, o SO interrompe a execução do PROG_1 e inicia execução das instruções do PROG_2. O mesmo ocorre posteriormente com PROG_2, que é interrompido para que PROG_3 seja executado. No instante t_{12} , o SO interrompe a execução das instruções do PROG_3 e retoma a execução do PROG_1, como se nenhuma interrupção houvesse acontecido. Essa troca de processos realizada pelo SO de forma transparente ao usuário é denominada **mudança de contexto** (MACHADO; MAIA, 2017).

Para que a concorrência ocorra de forma organizada, os processos assumem diferentes estados à medida que são executados. Assim, quando em execução, um processo pode se encontrar em um de três estados possíveis (MACHADO; MAIA, 2011; 2017):

- **Execução:** um processo encontra-se no estado de “execução” quando está sendo processado pela CPU. Em sistemas com apenas uma CPU, somente um processo é executado em um dado momento; em sistemas multiprocessados é possível mais de um processo ser executado ao mesmo tempo.

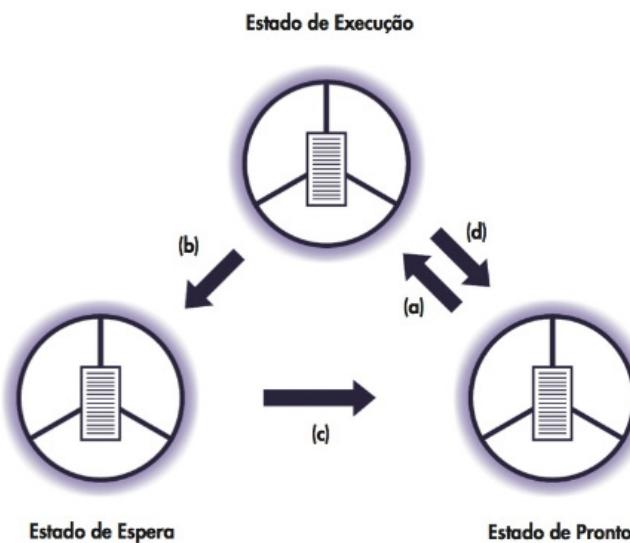
- **Pronto:** um processo encontra-se em estado de “pronto” quando está aguardando em uma fila à espera de sua execução. É o SO que estabelece a ordem de execução dos processos através de um mecanismo chamado escalonamento (sobre o qual falaremos na sequência).
- **Espera:** um processo encontra-se em estado de “espera” quando aguarda por algum recurso ou evento externo para continuar a ser executado.



Quando em estado de “pronto” um processo pode mudar para o estado de “execução”. Quando em estado de “execução” pode mudar para o estado de “pronto” ou para o estado de “espera”. Quando em estado de “espera” pode mudar para o estado de “pronto”.

A Figura 26 ilustra essas mudanças:

FIGURA 26 – MUDANÇAS DE ESTADO DE UM PROCESSO



FONTE: Machado e Maia (2017, p. 62)

“Um processo também pode ser definido como o ambiente onde um programa é executado” (MACHADO; MAIA, 2017, p. 63). Para permitir que um processo cumpra com sua função, este ambiente envolve informações sobre sua execução e recursos, como por exemplo, uso de CPU, espaço de memória, arquivos e dispositivos de E/S. Toda esta estrutura é fornecida quando o processo é criado (MACHADO; MAIA, 2011; 2017; SILBERSCHATZ; GALVIN; GAGNE, 2015).

Perceba que “um processo é formado por três partes, conhecidas como contexto de hardware, contexto de software e espaço de endereçamento, que juntos mantêm todas as informações necessárias à execução de um programa” (MACHADO; MAIA; 2017, p. 63). As partes que compõem um processo estão ilustradas na Figura 27.

FIGURA 27 – ESTRUTURA DO PROCESSO



FONTE: Machado e Maia (2017, p. 63)

Essa estrutura contendo as informações sobre o processo é chamada pelo SO de registro. Algumas das informações mantidas pelo registro de um processo envolvem (OLIVEIRA; CARISSIMI; TOSCANI, 2010):

- Prioridade do processo, que define a ordem de execução do processo pelo processador.
- Localização e tamanho da memória principal ocupada pelo processo.
- Identificação dos arquivos abertos no momento.
- Estado do processo: espera, execução ou pronto.
- Conteúdo dos registradores do processador quando o processo é suspenso temporariamente.

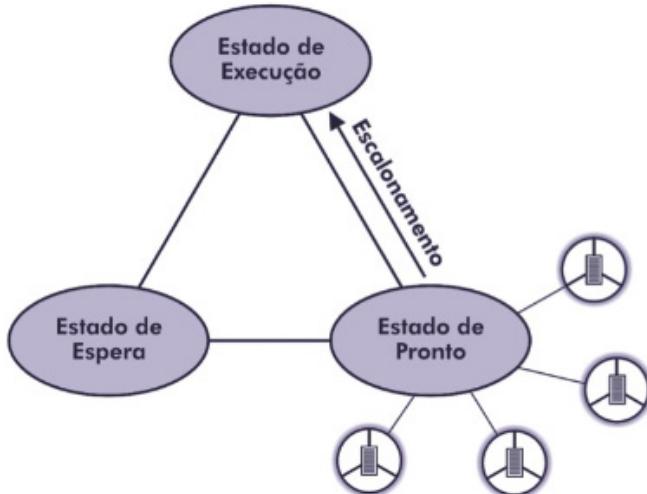
3 ESCALONAMENTO

Como você já sabe, o elemento chave relacionado aos SO multiprogramáveis está em sua capacidade de executar, concorrentemente, vários processos, maximizando assim o uso do processador e de outros recursos.

Você também viu que a concorrência entre os processos pelo uso do processador (especialmente quando só existe um no sistema computacional) é possível por causa do compartilhamento de tempo, alternando o uso entre os processos de tal modo que o usuário consegue interagir com todos os programas que está executando.

Nesta circunstância, a partir do momento em que mais de um processo encontra-se em estado de “pronto”, cabe ao SO definir qual será o primeiro a ser executado. A esta escolha sobre qual processo utilizará primeiro determinado recurso dá-se o nome de escalonamento, conforme mostra a Figura 28. Essa tarefa é feita pelo escalonador.

FIGURA 28 – ESCALONAMENTO



FONTE: Machado e Maia (2017, p. 127)



“Os critérios utilizados para esta seleção compõem a chamada política de escalonamento, que é a base da gerência do processador e da multiprogramação em um sistema operacional” (MACHADO; MAIA, 2017, p. 127).

O que determinará a política de escalonamento é o conjunto de características do SO, ou seja, uma política de escalonamento está diretamente ligada ao propósito do SO. Por exemplo, sistemas de tempo compartilhado apresentam critérios de escalonamento diferentes de sistemas de tempo real.

Veja agora alguns dos critérios utilizados em uma política de escalonamento para o gerenciamento de processos. Vale ressaltar que, de maneira geral, a maioria das políticas busca otimizar o uso do processador e o *throughput* (MACHADO; MAIA, 2011; 2017).

- **Utilização do processador:** é desejável que o processador permaneça a maior parte do tempo ocupado.
- **Throughput:** número de processos executados em um determinado espaço de tempo.
- **Tempo de processador:** tempo que um processo leva em estado de “execução”.
- **Tempo de espera:** tempo total que processo espera na fila em estado de “pronto”.
- **Tempo de turnaround:** tempo desde a criação até o término do processo.
- **Tempo de resposta:** tempo decorrido desde a requisição ao sistema até a exibição de sua resposta.



O escalonamento é exigido quando:

- Um processo termina.
- Um processo é bloqueado em uma operação de E/S.

Agora que você já sabe o que é o escalonamento de processos, é importante, também, que você saiba que existem dois tipos de escalonamento que podem ser realizados pelo SO: o escalonamento de curto prazo ou preemptivo, e o escalonamento de longo prazo ou não preemptivo.

- **Escalonamento preemptivo:** caracterizado por ser rápido, o escalonamento preemptivo seleciona entre os processos em estado de “pronto” e aloca o processador a um deles. É possível que exista priorização na execução dos processos, como em aplicações de tempo real, por exemplo, nas quais o tempo de resposta é crítico.
- **Escalonamento não preemptivo:** foi o primeiro tipo de escalonamento implementado. Era muito usual em sistemas *batch*, quando são encaminhados lotes de processos para execução. Logo, neste tipo de escalonamento, quando um processo está em execução, nenhum evento externo pode ocasionar a perda do uso do processador.

4 MEMÓRIA E ARMAZENAMENTO

Softwares, aplicativos ou de sistemas, executando sobre um sistema computacional manipulam dados, e alguns manipulam grandes volumes, utilizando, para isso, a memória principal para carregar e executar os seus processos e o sistema de armazenamento secundário para permitir gravar e, se necessário, recuperar esses dados sempre que necessário.

Dessa forma, um SO deve garantir a gerência da memória existente no sistema de forma a potencializar a quantidade de aplicações de usuários, utilizando adequadamente a quantidade de memória principal disponível. Paralelo a isso, o SO precisa ser capaz de manipular e organizar esses dados através da implementação de um sistema de armazenamento baseado em arquivos.

Você percebeu que os softwares ao serem carregados na memória principal de um computador precisam ter seus dados gravados permanentemente em estruturas que poderão ser recuperadas para futuras operações? Vamos ver, agora, algumas funções básicas da memória principal e de um sistema de armazenamento.

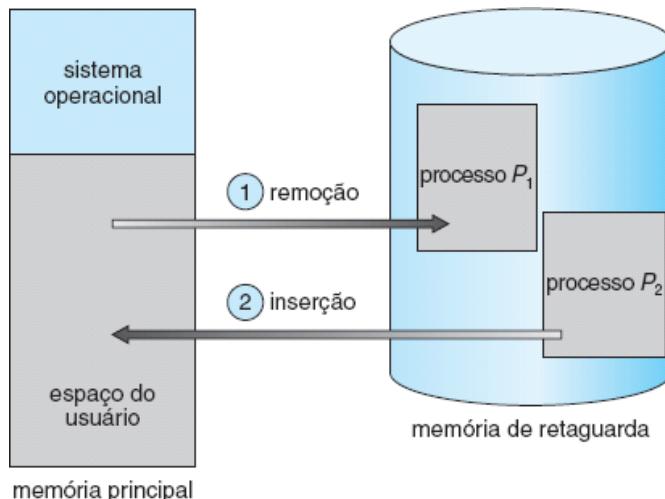
Em um sistema computacional bem gerenciado, um SO precisa manter na memória principal o maior número de processos residentes, além de maximizar o compartilhamento do processador e demais recursos pertinentes ao sistema, e que mesmo na total ocupação do espaço da memória o SO precisa garantir que novos processos sejam criados e executados. Uma outra condição que o SO precisa lidar é que mesmo uma única aplicação pode conter um processo maior que a capacidade da memória principal, e, nesta situação, parte da memória secundária é usada como uma espécie de “memória virtual” para armazenar o referido processo ou permitir a criação de novos (MACHADO; MAIA, 2017).

Além da gerência de memória, o SO precisa também gerenciar um sistema de armazenamento de arquivos, uma vez que é esse sistema o responsável pela manipulação constante que os usuários fazem dos arquivos físicos existentes em um sistema computacional (MACHADO; MAIA, 2017). Vamos então discutir um pouco mais sobre esses dois importantes elementos gerenciados pelo SO.

4.1 MEMÓRIA PRINCIPAL

O SO oferece formas de gerenciamento para este tipo de memória, sendo uma delas a técnica de permuta entre processos (*swapping*), que permite a transferência temporária para uma memória secundária e depois a recuperação para prosseguir com sua execução na memória principal, como mostrado na Figura 29.

FIGURA 29 – TROCA DE PROCESSOS ENTRE A MEMÓRIA PRINCIPAL E UMA MEMÓRIA SECUNDÁRIA



FONTE: Silberschatz, Galvin e Gagne (2015, s. p.)

Perceba que na Figura 29 o SO está removendo um processo (P_1) da memória principal e inserindo um outro processo (P_2), que estava armazenado em uma memória secundária (também chamada de memória de retaguarda) para que seja executado pelo processador. Isso permite que o mapeamento de endereços físicos de processos em execução exceda a quantidade de espaço para endereços físicos na memória principal, favorecendo a característica de multiprogramação de um sistema (SILBERSCHATZ; GALVIN; GAGNE, 2015).

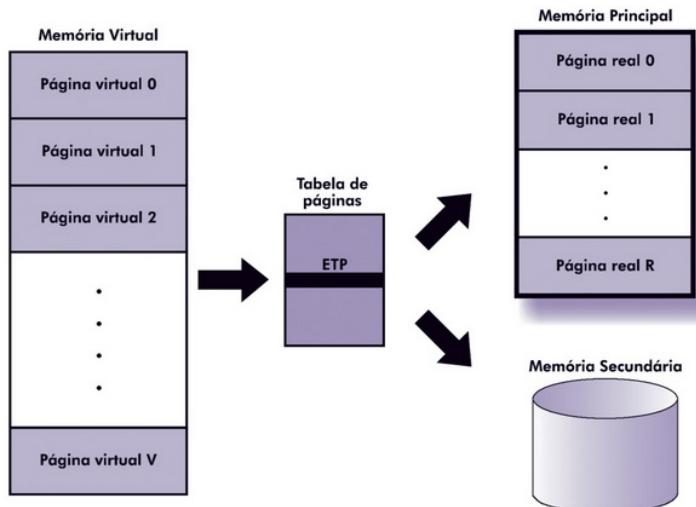
4.2 MEMÓRIA VIRTUAL

Memória virtual é uma técnica sofisticada e poderosa de gerência de memória, em que as memórias principal e secundária são combinadas dando ao usuário a ilusão de existir uma memória muito maior que a capacidade real da memória principal. O conceito de memória virtual fundamenta-se em não vincular o endereçamento feito pelo programa dos endereços físicos da memória principal. Desta forma, programas e suas estruturas de dados deixam de estar limitados ao tamanho da memória física disponível, pois podem possuir endereços associados à memória secundária (MACHADO; MAIA, 2017, p. 159).

Uma dessas técnicas é representada pela **memória virtual por paginação** na qual o endereçamento virtual (memória secundária) e o espaço de endereçamento real (memória principal) representam blocos do mesmo tamanho chamados de **páginas**: no domínio virtual elas se denominam **páginas virtuais**, já no espaço real, denominam-se **páginas reais ou frames** (MACHADO; MAIA, 2017).

Para o mapeamento são utilizadas tabelas de páginas, sendo cada processo detentor de sua própria tabela e com cada página do processo tendo uma entrada na tabela chamada de ETP (entrada na tabela de páginas), contendo informações de mapeamento apontando para a página real correspondente (MACHADO; MAIA, 2017). Veja como funciona a memória virtual por paginação na Figura 30.

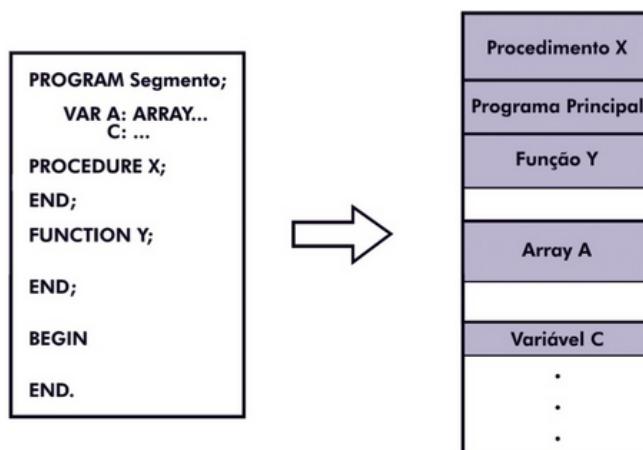
FIGURA 30 – MEMÓRIA VIRTUAL POR PAGINAÇÃO



FONTE: Machado e Maia (2017, p. 164)

Outra técnica apresentada por Machado e Maia (2017) é a **memória virtual por segmentação**; nela o endereçamento virtual se divide em blocos de tamanhos diferentes, os **segmentos**. Nesta técnica, um programa é dividido logicamente em sub-rotinas e estruturas de dados alocadas em segmentos na memória principal, conforme você pode observar na Figura 31.

FIGURA 31 – MEMÓRIA VIRTUAL POR SEGMENTAÇÃO



FONTE: Machado e Maia (2017, p. 181)

A técnica de segmentação, ao contrário de sua outra irmã, mantém uma relação entre a lógica do programa e sua alocação na memória principal. A criação dos segmentos, geralmente, é realizada pelo compilador, nos quais cada um pode representar um procedimento, função, vetor ou pilha (MACHADO; MAIA, 2017).

O casamento das duas técnicas gera a abordagem de memória virtual baseada por **segmentação com paginação**. Essa técnica, define um endereço virtual formado pelo número do segmento virtual (NSV), um número de página virtual (NPV) e um deslocamento. O NSV aponta para uma entrada na tabela de segmentos com informações da tabela de páginas do segmento. Já o NPV identifica, exclusivamente, a página virtual com o endereço (índice na tabela de páginas). O deslocamento fornece a posição do endereço virtual considerando o início da página na qual se encontra, logo, o endereço físico é obtido através do endereço do *frame* (localizado na tabela de páginas), mais o deslocamento (contido no endereço virtual) (MACHADO; MAIA, 2017).

4.3 ESTRUTURA DE ARMAZENAMENTO

Um arquivo é constituído por informações (instruções ou dados) logicamente relacionadas. Por exemplo, arquivos executáveis contém instruções compreendidas pela CPU, já um arquivo de dados pode ser representado como um arquivo-texto (uma planilha por exemplo) ou de modo mais formal, como um conjunto de linhas e colunas em um banco de dados relacional. São inúmeras as formas de armazenar um arquivo: fitas magnéticas, discos magnéticos e discos ópticos, e os mesmos são diferenciados pela sua extensão, conforme apresenta o Quadro 4. Sendo que estes dispositivos de armazenamento devem ser isolados pelo sistema operacional (MACHADO; MAIA, 2017).

QUADRO 4 – EXTENSÃO DE ARQUIVOS

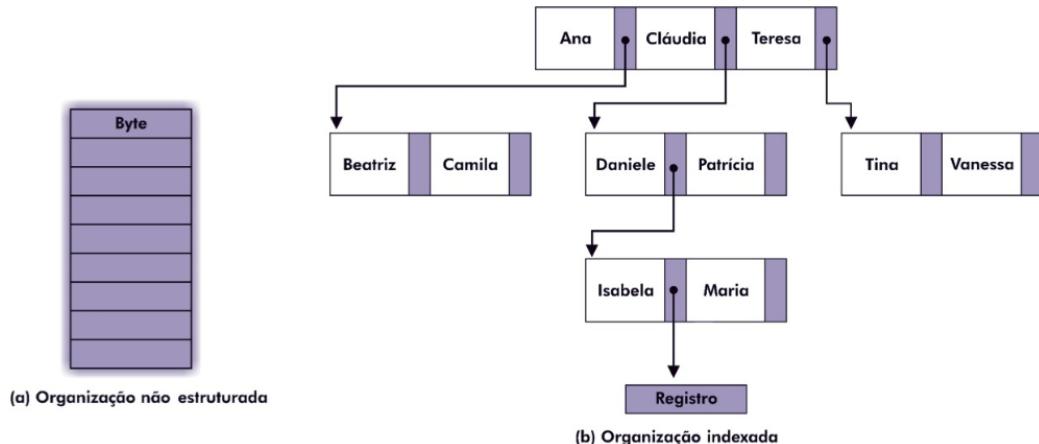
Extensão	Descrição
Avaliação I.docx	Arquivo padrão do MS Word
Programa.java	Arquivo fonte padrão da linguagem de programação Java
Index.html	Arquivo fonte padrão no formato de arquivos interpretados pelos softwares navegadores
firefox.exe	Arquivo executável que inicia o navegador Mozilla Firefox.

FONTE: Os autores

Nestes arquivos a organização diz respeito em como os seus dados estão internamente armazenados e a estrutura dos seus dados, que pode variar em função do tipo de informação contida no arquivo. Perceba, arquivos-textos (programa.java) possuem aplicação completamente diferente de arquivos executáveis (firefox.exe), possuindo, consequentemente, estruturas diferentes. A Figura 32(a), mostra a forma mais simples de organização de arquivos, através

de uma sequência não estruturada de bytes. Neste tipo de organização não existe uma estrutura lógica para os dados, quem define isso é a aplicação. De outro modo, a Figura 32(b) mostra o modo de organização indexado. Neste caso, os arquivos são organizados em registros, e cada registro possui um índice que aponta para o registro seguinte (MACHADO; MAIA, 2017).

FIGURA 32 – ORGANIZAÇÃO DE ARQUIVOS



FONTE: Machado e Maia (2017, p. 195)



Para complementar seus estudos deste tópico, acesse a Minha Biblioteca (<https://bibliotecavirtual.uniassselvi.com.br/>) e leia os subcapítulos 8.5, 8.6 e 8.7 do livro *Arquitetura de sistemas operacionais*, de Francis Berenger Machado e Luiz Paulo Maia para conhecer sobre outros tipos de escalonamento de processos.



A Leitura Complementar selecionada para você nesta unidade traz um pouco mais sobre os SO móveis iOS e Android. O material completo com vídeos e imagens encontra-se disponível em: <https://www.tecmundo.com.br/samsung/3702-sistemas-operacionais-moveis-qual-a-diferenca-.htm>.

LEITURA COMPLEMENTAR

SISTEMAS OPERACIONAIS MÓVEIS: QUAL A DIFERENÇA?

Camila Camargo

Não há como negar, os telefones celulares, smartphones e dispositivos móveis ocupam cada vez mais tempo e espaço em nossas vidas. Seus processadores estão mais velozes, há mais memória e um salto considerável no que diz respeito ao armazenamento foi dado.

Praticamente tudo o que antes era possível apenas pelos computadores de mesa agora pode ser feito em qualquer lugar, a qualquer hora do dia, direto da palma da sua mão. Junto à evolução, chegaram as empresas querendo “ganhar o seu” na guerra da mobilidade.

Nokia, Google, Apple, Microsoft, Samsung, Intel e companhia possuem sistemas operacionais próprios. Isso é legal, pois estimula a concorrência, porém a quantidade de opções e modelos que invadem as tabelas de configurações certamente dão um nó na cabeça de muita gente.

Para ajudá-lo, neste artigo você encontra uma breve descrição dos principais sistemas operacionais para celulares presentes no mercado, justamente para que possa entender as diferenças e semelhanças entre eles na hora de escolher um novo aparelho.

Para começar, vamos ao SO que revolucionou o mundo dos celulares e trouxe uma nova visão para o mercado:

Mac OSX – iPhones

O que faz com que o iPhone, o iPod Touch e o iPad rodem com tamanha maestria é uma versão modificada do sistema operacional Mac OSX, que recebe o nome de iPhoneOS. Seu foco é oferecer suporte para as tecnologias de reconhecimento de toques múltiplos, de inclinação (graças à inclusão do acelerômetro interno) e de multimídia, para a reprodução de vídeos, imagens e músicas.

A interface é simplificada, composta por ícones espalhados na área principal e outros fixos na parte de baixo da tela, chamada de “Dock”, para ligação, mensagens, e-mail e outros de sua preferência. Para trocar de “telas”, basta arrastar o dedo de um lado para o outro, ao passo que para abrir o aplicativo é necessário apenas um toque sobre seu ícone.

No dia 3 de fevereiro a Apple lançou a versão 3.1.3 do sistema operacional que contém três melhorias pouco significativas. Espera-se que em breve a versão 3.2 chegue e seja compatível com o iPhone, pois parece que será exclusiva ao iPad. Apesar da interface impecável, há quem torça o nariz para as limitações impostas pela fabricante.

Você só pode baixar os aplicativos disponíveis na AppleStore — a menos que recorra à liberação do aparelho para aplicativos não oficiais, mas perca a garantia —, o que possui pontos bons e ruins. O lado positivo é que você sabe que tudo o que você baixar vai funcionar. Vírus ou outras ameaças com certeza não chegarão ao seu aparelho.

O lado ruim é a dependência da loja oficial e as restrições de compra de alguns aplicativos. Nem sempre é possível comprar o que você deseja na loja norte-americana, por exemplo. Também não há suporte para conectividade com dispositivos de terceiros, limitando o acesso a componentes Bluetooth.

Android

O SO da gigante Google (que conta com um consórcio de mais de 34 empresas para bancá-lo) é também o que mais causa alarde na indústria de sistemas para portáteis e celulares, justamente por ter um apoio tão forte e pela sua natureza, de programação aberta, acessível a todos os interessados.

Baseado em Linux, ele traz consigo suporte para todo tipo de conexão sem fio (3G, EDGE, Wi-Fi e Bluetooth), para multimídia — inclusive vídeos de alta definição — e é extremamente versátil, facilmente adaptado a PDAs ou aos tradicionais telefones em barra, com suas telas menores.

Há menos de um ano no mercado, o Android é a bola da vez no mercado de celulares. Várias empresas anunciaram recentemente a utilização do sistema operacional para equipar seus aparelhos. A Motorola, Acer, HTC, Sony Ericsson e HP, além do próprio Google, são alguns exemplos de empresas que adoram o SO.

A sua grande capacidade de modificação, adaptação e o seu custo baixíssimo o tornam uma excelente escolha de sistema para aparelhos um pouco mais robustos, levando a antiga batalha contra o Windows a novos territórios.

Você encontra-o em aparelhos como Motorola ROKR E8 e RAZR 2 V8, Samsung, Nokia N810, HTC Hero e Nexus One. De acordo com a Google, a versão 1.6 representa 47,6% do total de aparelhos com o sistema operacional. A mais recente, a 2.1, já detém 20,4%.

FONTE: CAMARGO, C. Sistemas operacionais móveis: qual a diferença? **Tecmundo**, Curitiba, fev. 2010. Disponível em: <https://www.tecmundo.com.br/samsung/3702-sistemas-operacionais-moveis-qual-a-diferenca-.htm>. Acesso em: 15 jul. 2019.

RESUMO DO TÓPICO 5

Neste tópico, você aprendeu que:

- Um processo pode ser entendido como um programa em execução e é considerado a unidade de trabalho em um SO de tempo compartilhado.
- O gerenciamento de processos envolve tarefas como criar e excluir processos de usuário e do sistema, suspender e retomar processos, fornecer mecanismos de sincronização de processos e fornecer mecanismos de comunicação entre processos.
- Os processos assumem diferentes estados enquanto são executados. Esses estados são chamados de: “execução”, “pronto” e “espera”.
- Escalonamento é a função responsável pela escolha sobre qual processo utilizará primeiro determinado recurso. O escalonamento é realizado pelo escalonador.
- A política de escalonamento estabelece os critérios utilizados para a seleção dos processos.
- Existem dois tipos de escalonamento: preemptivo e não preemptivo.
- Para a manipulação de grandes volumes de dados o computador se utiliza da memória principal.
- Um SO precisa manter na memória principal o maior número de processos residentes, gerenciando, além da memória, um sistema de armazenamento de arquivos.
- O *swaping* é uma técnica de permuta entre processos, transferindo-os da memória principal para uma memória secundária.
- Memória virtual é uma técnica sofisticada e poderosa de gerência de memória, em que as memórias principal e secundária são combinadas dando, ao usuário, a ilusão de existir uma memória muito maior que a capacidade real da memória principal.



Ficou alguma dúvida? Construímos uma trilha de aprendizagem pensando em facilitar sua compreensão. Acesse o QR Code, que levará ao AVA, e veja as novidades que preparamos para seu estudo.



AUTOATIVIDADE



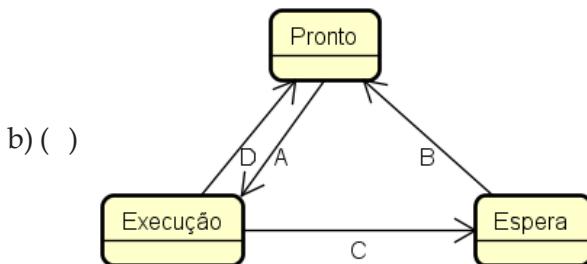
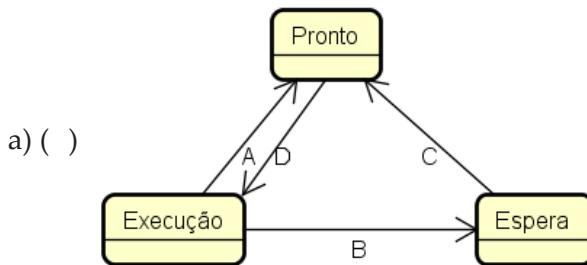
1 É definido como processo um programa que se encontra em execução. Em um sistema computacional multiprogramável ou multitarefa, vários processos podem ser acionados, simultaneamente, pelo usuário. Assim, para que a concorrência ocorra de forma organizada, os processos assumem diferentes estados à medida que são executados, que são: execução, pronto e espera. A partir disso, avalie os seguintes cenários:

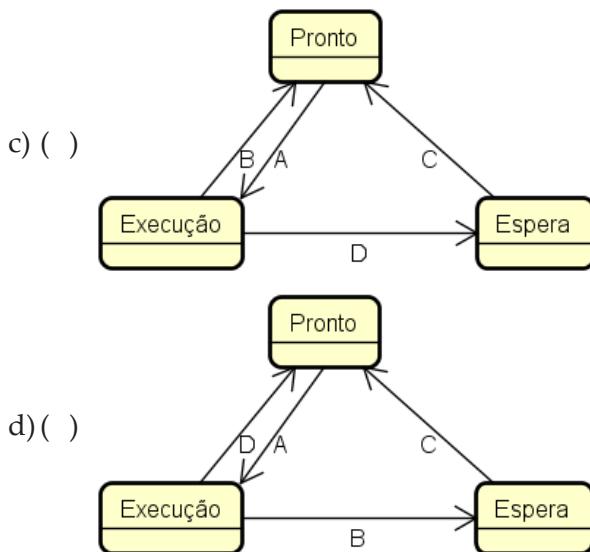
- I- Um documento é encaminhado para a fila de impressão.
- II- Um software aplicativo de e-mail é acionado pelo usuário.
- III- Um sistema aguarda pela confirmação do usuário para finalização.

Os cenários apresentados nas assertivas se encontram em que estados, respectivamente?

- a) () Execução; pronto; espera.
- b) () Execução; execução; pronto.
- c) () Espera; pronto; execução.
- d) () Pronto; execução; espera.

2 É definido como processo um programa que se encontra em execução. Em um sistema computacional multiprogramável ou multitarefa, vários processos podem ser acionados, simultaneamente, pelo usuário. Assim, para que a concorrência ocorra de forma organizada, os processos assumem diferentes estados à medida que são executados, que são: execução, pronto e espera. A partir disso, avalie as figuras e indique aquela que pode representar as transições entre os estados de um processo:





3 Pode-se considerar que um processo pode ser visualizado em três dimensões distintas, mas que se complementam para a sua composição. Cada uma delas define o que se chama de contexto, isto é, mantém à sua volta partes de elementos que se ligam para compor um todo, o processo. Com isso em mente, avalie as sentenças e indique aquela que contém essas dimensões:

- a) () Espaço de armazenamento e contextos físico e de software.
- b) () Espaço de endereçamento e contextos de hardware e de sistema.
- c) () Espaço de endereçamento, contextos de hardware e de software.
- d) () Espaço de armazenamento e contextos de hardware e lógico.

4 Pode-se considerar que um processo pode ser visualizado em três dimensões distintas, mas que se complementam para a sua composição. Cada uma delas define o que se chama de contexto, isto é, mantêm à sua volta partes de elementos que se ligam para compor um todo, o processo, sendo que essas dimensões registram certas informações de um processo. Sabendo disso, avalie as sentenças e indique aquelas que representam as informações mencionadas:

- a) () Dados dos registradores do processador, prioridade e estado do processo, informações da memória principal usada pelo processo e indicação de arquivos abertos.
- b) () Dados dos registradores na memória principal, prioridade do processo, informações da memória principal usada pelo processo e indicação de arquivos abertos.
- c) () Dados dos registradores da memória, estado do processo do processo, informações da memória principal usada pelo processo e indicação de arquivos abertos.
- d) () Dados dos registradores do processador, prioridade e estado do processo, informações da memória secundária usada pelo processo e indicação de arquivos abertos.

- 5 Um SO não executa aleatoriamente os seus processos, mas aqueles processos que se encontram em estado de “pronto” são elegíveis para execução baseados em critérios regidos pela sua política de escalonamento no seu gerenciamento de processos. Desta forma, tendo em mente estes critérios, assinale a resposta INCORRETA sobre os mesmos:
- a) () Tempo de processador representa o tempo no qual um processo executa no processador.
 - b) () *Throughput* representa a quantidade de processos executados considerando um espaço de tempo especificado.
 - c) () Utilização do processador significa que se espera que o processador se mantenha a maior parte do tempo desocupado.
 - d) () Tempo de espera significa o total de tempo em estado de pronto para execução que um processo aguarda na fila.

UNIDADE 2

SISTEMAS DISTRIBUÍDOS

OBJETIVOS DE APRENDIZAGEM

A partir do estudo desta unidade, você será capaz de:

- conhecer as principais características e conceitos de sistemas distribuídos;
- compreender a arquitetura dos sistemas distribuídos e algumas de suas classificações existentes;
- entender o conceito cliente/servidor e chamadas de procedimentos remotos;
- compreender a relação do conceito de processos e threads no contexto dos sistemas distribuídos;
- compreender as principais características e conceitos da computação móvel e ubíqua.

PLANO DE ESTUDOS

Esta unidade está dividida em cinco tópicos. No decorrer da unidade você encontrará autoatividades com o objetivo de reforçar o conteúdo apresentado.

TÓPICO 1 – CONCEITOS BÁSICOS SOBRE SISTEMAS DISTRIBUÍDOS

TÓPICO 2 – ARQUITETURA PARA SISTEMAS DISTRIBUÍDOS

TÓPICO 3 – PROCESSOS

TÓPICO 4 – TIPOS DE SISTEMAS DISTRIBUÍDOS

TÓPICO 5 – COMPUTAÇÃO MÓVEL E UBÍQUA



Preparado para ampliar seus conhecimentos? Respire e vamos em frente! Procure um ambiente que facilite a concentração, assim absorverá melhor as informações.

CONCEITOS BÁSICOS SOBRE SISTEMAS DISTRIBUÍDOS

1 INTRODUÇÃO

Na sua trajetória pela Unidade 1, você se dedicou a estudar sobre os SO. A partir dos conteúdos estudados, você agora já comprehende que o SO é um tipo de software muito específico, responsável por realizar a interface entre o usuário e os softwares aplicativos, e entre estes e os demais recursos de um sistema computacional.

Compreendemos que, ao longo das últimas décadas, dada sua finalidade, os SO evoluíram em diferentes aspectos, desde a forma de realização de processamento até, especialmente, o desenvolvimento de interfaces mais simples e eficientes. Neste último caso, pode-se afirmar, sem exageros, que os SO existem para facilitar a vida do usuário, simplificando a utilização dos diversos sistemas computacionais, desde os PCs, passando pelos notebooks, e até mais recentemente os *smartphones*.

Não podemos, no entanto, nos esquecer de que, se os SO são convenientes do ponto de vista de seu uso, facilitando a realização de tarefas que antes exigiam conhecimentos específicos, devem também ser eficientes no tocante às inúmeras e complexas funções que executam. E como já comentamos antes na Unidade 1, para cada finalidade há um tipo específico de SO a ser implementado.

E é exatamente sobre um tipo específico de SO que trataremos de estudar a partir desta Unidade 2, os chamados sistemas distribuídos. Você deve estar lembrado que falamos rapidamente sobre os sistemas distribuídos no Tópico 4 da Unidade 1.

Então, para iniciarmos nossos estudos sobre os sistemas distribuídos, é importante relembrarmos de seu conceito, juntamente de outro conceito muito importante, o conceito de SO de redes, que passou a ser aplicado a partir da década de 1980, com o advento das redes de computadores.

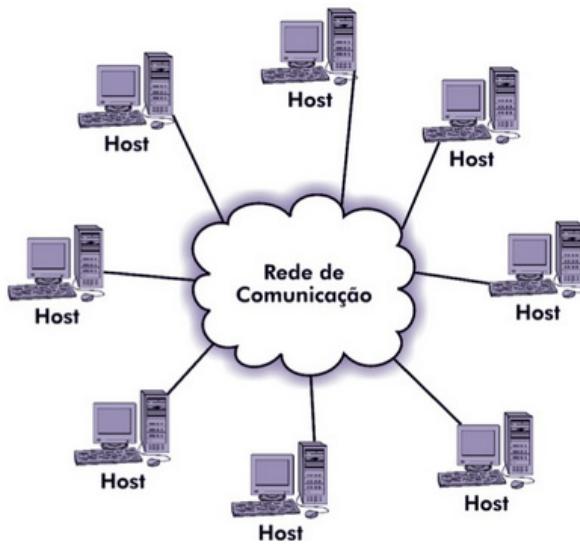
Não podemos esquecer que “com a evolução dos computadores pessoais e das estações de trabalho, juntamente com o avanço das telecomunicações e da tecnologia de redes, surgiu um novo modelo de computação, chamado modelo de rede de computadores” (MACHADO; MAIA, 2017, p. 21). Antes das redes de computadores tinha-se a disposição de sistemas de grande porte que realizavam o processamento das informações de forma centralizada, já que os terminais de acesso ao usuário não possuíam qualquer capacidade de processamento (eram os chamados “terminais burros”, lembra-se?).



Nos modelos de processamento centralizado, sempre que o usuário deseja alguma coisa, uma requisição é enviada a partir de seu terminal ao sistema localizado no servidor, por exemplo, que realiza o processamento e retorna uma resposta (MACHADO; MAIA, 2011; 2017).

Diferentemente dos antigos sistemas centralizados, em uma rede de computadores existem dois ou mais sistemas computacionais distintos interligados por intermédio de linhas de comunicação, capazes de trocar informações entre si, sem a necessidade de um sistema centralizado para intermediar estes serviços (MACHADO; MAIA, 2011; 2017), conforme pode ser visto na Figura 1.

FIGURA 1 – REDE DE COMPUTADORES



FONTE: Machado e Maia (2017, p. 235)



Os sistemas computacionais independentes que compõem uma rede de computadores são popularmente conhecidos como *hosts*.

A evolução das redes de computadores permitiu que elas crescessem em número de *hosts*, exigindo projetos de sistemas mais complexos que atuassem com maior velocidade e confiabilidade. Como resultado, tecnologias foram desenvolvidas e aprimoradas para permitir que grandes quantidades de máquinas, localizadas em um edifício, por exemplo, ou distribuídas em espaços geográficos mais distantes, fossem conectadas por estruturas de alta velocidade (DEITEL; DEITEL; CHOHNES, 2005; TANENBAUM; STEEN, 2007).

Desde esta revolução, na forma de processar e distribuir informações, as redes de computadores estão por toda parte, como em redes corporativas, em fábricas, redes nos campus das universidades e, mais recentemente, as redes de telefones móveis. Não podemos também nos esquecer da mais conhecida e utilizada entre elas, a internet, e as muitas redes que a compõe (COULOURIS *et al.*, 2013).

A partir disso, como comentado no início deste item, o entendimento das redes de computadores, seu funcionamento e características, tornam-se importantes para o estudo dos diferentes tipos de SO e sua utilização. A existência de vários *hosts* em uma rede de computadores a torna um ambiente de SO multiprocessado. Logo, é baseado no grau de integração dos *hosts* da rede, que torna possível identificar a necessidade de um SO fricamente acoplado, que pode ser de (pelo menos) dois tipos: os SO de rede e os sistemas distribuídos (Figura 2).

FIGURA 2 – TIPOS DE SO MULTIPROCESSADOS FRACAMENTE ACOPLADOS



FONTE: Adaptado de Machado e Maia (2017, p. 225)

É sobre a definição e finalidade dos sistemas distribuídos que nos dedicaremos nos próximos itens.



Lembre-se que ao nos referirmos a sistemas distribuídos estamos falando de sistemas operacionais distribuídos. Trata-se de sinônimos, e adotaremos ao longo do livro o termo "sistemas distribuídos" ao invés de SO distribuídos, apenas por uma questão de convenção da literatura.

2 O QUE SÃO SISTEMAS DISTRIBUÍDOS?

Você acabou de relembrar que os sistemas distribuídos são um tipo específico de SO, os SO fricamente acoplados, cujo funcionamento está relacionado à capacidade de multiprocessamento a partir de diversos *hosts* integrados em uma estrutura de rede. Porém, perceba que esta é uma classificação dos sistemas distribuídos, mas ainda não estabelecemos sua definição, ou seja, o que de fato são sistemas distribuídos.

Várias são as definições encontradas na literatura que estabelecem o que é um sistema distribuído. É possível encontrarmos definições bem simples e diretas, como as seguintes:

- “Um sistema distribuído é um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente” (TANENBAUM; STEEN, 2007, p. 1).
- “Um sistema distribuído é aquele no qual os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas passando mensagens” (COULOURIS *et al.*, 2013, p. 1).

Ou você poderá encontrar ainda definições mais elaboradas, com um nível maior de detalhes como esta:

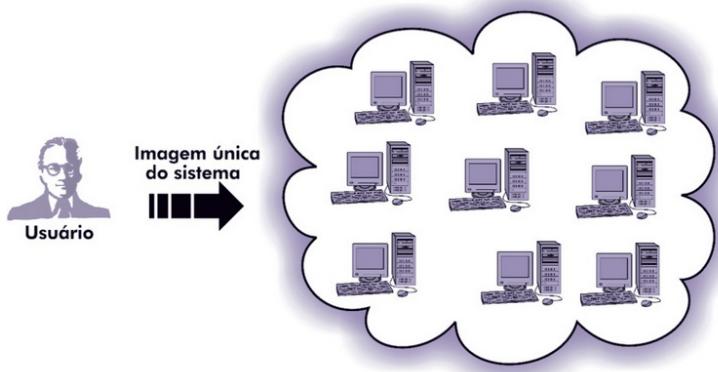
Um sistema distribuído é um conjunto de sistemas autônomos, interconectados por uma rede de comunicação e que funciona como se fosse um sistema fortemente acoplado. Cada componente de um sistema distribuído possui seus próprios recursos, como processadores, memória principal, dispositivos de E/S, sistema operacional e espaço de endereçamento. Os tipos de sistemas operacionais que compõem um sistema distribuído não precisam ser necessariamente homogêneos (MACHADO; MAIA, 2017, p. 237).

Independentemente da definição a ser adotada ou daquela que mais lhe agrada, perceba que todas elas, com palavras diferentes, expressam alguns elementos em comum: computadores independentes ou autônomos, interligação em rede, troca de informações (mensagens), sistema único.

Desta forma, podemos, então, formalizar uma nova definição que compreenda todos esses elementos para facilitar ainda mais seu entendimento sobre sistemas distribuídos! Nesta nossa definição, podemos dizer que sistemas distribuídos são formados por sistemas computacionais independentes e autônomos, interligados entre si por meio de uma estrutura de rede. Estes computadores são capazes de se comunicar e colaborar entre si através desta estrutura, passando ao usuário a ideia de um sistema único (COULOURIS *et al.*, 2013; TANENBAUM; STEEN, 2007).

A Figura 3 demonstra o conceito de sistemas distribuídos do ponto de vista do usuário: para ele é como se não houvesse uma rede de computadores, mas um único sistema (MACHADO; MAIA, 2011; 2017):

FIGURA 3 – SISTEMA DISTRIBUÍDO



FONTE: Machado e Maia (2017, p. 237)

Repare que, do ponto de vista de hardware, em um sistema distribuído, existem vários sistemas computacionais distintos, cada qual com seus recursos como memória e processador. A ilusão de que existe um único computador acontece porque existe um único SO gerenciando tais recursos. Desta forma, é possível que um determinado processo, independentemente de sua localização dentro da rede, acesse todos os recursos do sistema (DEITEL; DEITEL; CHOHNES, 2005; SILBERSCHATZ; GALVIN; GAGNE, 2015).

Em outras palavras, o que acontece é que em um sistema distribuído, uma aplicação, como, por exemplo, rodar a folha de pagamento de centenas de colaboradores de uma grande indústria, seja dividida em partes que serão executadas por diferentes *hosts* em uma grande rede de computadores. Para o usuário tudo acontece de forma transparente, como se o que existisse fosse um único sistema (MACHADO; MAIA, 2011; 2017). Por esta razão, sistemas distribuídos são usualmente complexos em sua implementação.



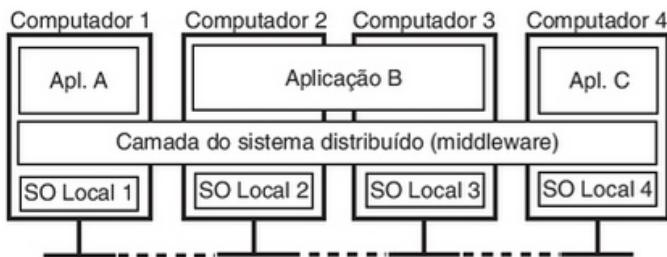
Tanto os sistemas distribuídos quanto os sistemas de rede estendem computação e armazenamento por toda uma rede de computadores [...]. Aplicações de sistemas distribuídos podem executar código em máquinas locais e remotas e compartilhar dados, arquivos e outros recursos entre máquinas (DEITEL; DEITEL; CHOHNES, 2005, p. 507).

No tocante à sua implementação, sistemas distribuídos são comumente chamados *middleware*, por representarem uma camada intermediária entre os SO locais de cada um dos computadores e as aplicações utilizadas pelos usuários.

Para suportar computadores e redes heterogêneos e, simultaneamente, oferecer uma visão de sistema único, os sistemas distribuídos costumam ser organizados por meio de uma camada de software - que é situada logicamente entre uma camada de nível mais alto, composta de usuários e aplicações, e uma camada subjacente, que consiste em sistemas operacionais [...] (TANENBAUM; STEEN, 2007, p. 2).

A Figura 4 mostra um exemplo do comportamento do sistema distribuído que, conforme ilustrado, é o responsável por distribuir uma mesma aplicação (Aplicação B) entre dois componentes da rede (Computador 2 e Computador 3).

FIGURA 4 – CAMADA DE SISTEMA DISTRUIÓDO (MIDDLEWARE)



FONTE: Tanenbaum e Steen (2017, p. 2)

3 PROPÓSITO

Agora que você já entendeu o que é um sistema distribuído chegamos ao momento de compreender seu propósito, ou seja, responder àquela segunda pergunta “para que serve?”.

Em se tratando de sistemas distribuídos, a resposta a esta pergunta está relacionada basicamente a dois aspectos. O primeiro deles remonta a própria evolução dos SO (sobre a qual comentamos no Tópico 2 da Unidade 1), para sistemas que pudesse executar vários processos simultaneamente, tornando-se, assim, eficientes. O segundo aspecto diz respeito ao surgimento das redes de computadores, cuja interligação entre sistemas computacionais permitiu uma nova forma de comunicação.

Desta maneira, pode-se afirmar que o desenvolvimento de um sistema distribuído está relacionado à melhoria. Melhoria da capacidade de processamento de um sistema computacional, de sua capacidade de armazenamento e, também, à possibilidade de prover o compartilhamento de recursos de forma confiável (COULOURIS *et al.*, 2013; DEITEL; DEITEL; CHOFFNES, 2005).



"A principal meta de um sistema distribuído é facilitar aos usuários, e às aplicações, o acesso a recursos remotos e seu compartilhamento de maneira controlada e eficiente" (TANENBAUM; STEEN, 2007, p. 2).

Quando falamos em compartilhamento de recursos remotos, estamos nos referindo aos componentes de hardware, como impressoras e discos de armazenamento, e também de software, como arquivos, bancos de dados, páginas web, e até mesmo vídeos e conexões de áudio (COULOURIS *et al.*, 2013; DEITEL; DEITEL; CHOFFNES, 2005; TANENBAUM; STEEN, 2007).

Basicamente, o que justifica o desenvolvimento de sistemas que permitam o compartilhamento é a economia. Por exemplo, é muito mais barato compartilhar uma impressora entre os diversos colaboradores de um departamento em uma empresa, do que adquirir uma impressora para cada colaborador que utiliza um computador. Em outras palavras: "dividir os recursos entre um grupo de máquinas permite que vários usuários os compartilhem e, ao mesmo tempo, garante que haja recursos suficientes para um trabalho ocasional de grande porte" (DEITEL; DEITEL; CHOFFNES, 2005, p. 507), como, por exemplo, compartilhar recursos de supercomputadores ou sistemas de armazenamento de alto desempenho (TANENBAUM; STEEN, 2007).

"Uma outra razão para adotar sistemas distribuídos é atender a uma grande base de usuários" (DEITEL; DEITEL; CHOFFNES, 2005, p. 507). A possibilidade de conexão entre usuários dispersos geograficamente favorece a colaboração e a troca de informações. A melhor forma de retratar esta situação é a internet. A partir da conectividade promovida pela internet é possível, por exemplo, a realização dos chamados *groupwares*, ou seja, grupos de pessoas localizadas em lugares diferentes (cidades ou mesmo países) que podem trabalhar juntas a partir de softwares de colaboração ou pela realização de vídeo conferências, por exemplo (TANENBAUM; STEEN, 2007).



Um software de colaboração muito utilizado é o Google Docs. Esta ferramenta permite que vários usuários remotos editem um mesmo documento de texto simultaneamente.

Repare que, para que este processamento distribuído entre os vários *hosts* que compõem esse tipo de sistema ocorra adequadamente, há algumas características que lhes são intrínsecas, como, por exemplo: escalabilidade, concorrência entre os componentes do sistema, segurança, tratamento de falhas e transparência (COULOURIS *et al.*, 2013; MACHADO; MAIA, 2011; 2017; TANENBAUM; STEEN, 2007). Essas características garantem a qualidade no fornecimento dos serviços em um sistema distribuído e merecem ser compreendidas.

Uma vez que sistemas distribuídos são executados em ambientes de rede, sejam elas redes locais ou de longa distância, é de grande importância que eles sejam escaláveis. Falando-se de maneira bem simples, a **Escalabilidade** é a característica que permite ao sistema distribuído manter-se eficiente mesmo que novos componentes (recursos ou usuários) sejam acrescentados.

Contudo, a escalabilidade de um sistema distribuído não pode ser considerada somente em termos do aumento de seu tamanho (agregação de recursos e usuários). A escalabilidade também pode estar relacionada a um contexto geográfico (à medida que usuários e recursos estão distantes uns dos outros), e em contexto gerencial (deve ser facilmente gerenciável mesmo que envolva usuários e recursos de diferentes organizações) (NEUMAN, 1994 *apud* TANENBAUM; STEEN, 2007).

A fim de que se possa usufruir da escalabilidade no projeto de um sistema distribuído é importante observar os seguintes aspectos, listados no Quadro 1:

QUADRO 1 – DESAFIOS RELACIONADOS À ESCALABILIDADE DE SISTEMAS DISTRIBUÍDOS

Controlar o custo dos recursos físicos.	Os custos para satisfazer novas demandas por recursos devem ser razoáveis.
Controlar a perda de desempenho.	Para que um sistema seja escalável, a perda de desempenho não deve ser maior que a função do tempo de acesso aos dados do sistema.
Impedir que os recursos de software se esgotem	Prever antecipadamente as necessidades do sistema é uma tarefa complexa. Eventualmente, a adaptação às mudanças pode ser a melhor escolha.
Evitar gargalos de desempenho.	Para se evitar gargalos no sistema deve-se fazer uso de algoritmos descentralizados.

FONTE: Adaptado de Coulouris *et al.* (2013)

Outra característica a ser considerada nos sistemas distribuídos é sua capacidade de utilizar os diferentes recursos disponíveis concorrentemente. A **concorrência** em um ambiente de rede é algo natural, ou seja, a chance de vários usuários acessarem um mesmo recurso compartilhado é grande. Isso significa dizer que dois usuários distintos, cada um em seu computador, podem compartilhar arquivos, por exemplo, sempre que necessário. O processamento concorrente das solicitações dos usuários garante que seja mantido o desempenho do sistema distribuído, sem, no entanto, que os dados compartilhados percam sua consistência.

A próxima característica da qual falaremos figura entre as mais importantes e deve ser considerada exatamente como uma consequência da conectividade e do compartilhamento presentes nos sistemas distribuídos. Trata-se da **segurança**. Via de regra, segurança em um sistema distribuído não se trata apenas de [...] uma questão de ocultar o conteúdo das mensagens – ela também envolve saber com certeza a identidade do usuário, ou outro agente, em nome de quem uma mensagem foi enviada” (COULOURIS *et al.*, 2013, p. 19).



A segurança está ancorada em três pilares: (1) confidencialidade: garante que recursos e informações não serão expostos a pessoas não autorizadas; (2) integridade: garante que recursos e/ou informações não serão alterados indevidamente e/ou sem permissão; e (3) disponibilidade: garante que os recursos e/ou informações estarão disponíveis quando solicitados, sem interferências (COULOURIS *et al.*, 2013).

Considerada fator crítico para um sistema distribuído, a **transparência** é a característica que permite criar ao usuário a ideia de que todos os sistemas computacionais distribuídos, independentemente, formam um sistema único.

O conceito de transparência permite que o usuário tenha acesso às suas informações e demais recursos disponíveis, independentemente da localização física dos mesmos, e sem que haja a necessidade de especificar o nome do componente ao qual o usuário esteja conectado. Exatamente porque o processamento ocorre de forma transparente aos olhos do usuário, ele não saberá em quais, ou quantos, componentes do sistema sua aplicação foi distribuída e não terá conhecimento caso algum erro ocorra.

A transparência em um sistema distribuído é qualificada em alguns tipos, conforme pode ser verificado no Quadro 2:

QUADRO 2 – TIPOS DE TRANSPARÊNCIA EM SISTEMAS DISTRIBUÍDOS

Transparência	Descrição
Acesso	Oculta diferenças na representação de dados e no modo de acesso a um recurso.
Localização	Oculta o lugar em que um recurso está localizado.
Migração	Oculta que um recurso pode ser movido para outra localização.
Relocação	Oculta que um recurso pode ser movido para outra localização enquanto em uso.
Replicação	Oculta que um recurso é replicado.
Concorrência	Oculta que um recurso pode ser compartilhado por diversos usuários concorrentes.

Falha	Oculta a falha e a recuperação de um recurso.
Paralelismo	Possibilita que uma aplicação paralela seja executada em qualquer processador de qualquer sistema.
Desempenho	Oferece, aos usuários, tempo de resposta independente de alterações na estrutura do sistema ou na sua carga.
Escalabilidade	Permite que o sistema cresça sem a necessidade de alterar as aplicações e seus algoritmos.

FONTE: Adaptado de ISO (1995 *apud* TANENBAUM; STEEN, 2007, p. 3);
Machado e Maia (2017, p. 238)



A transparência de desempenho é especialmente importante já que [...] operações realizadas remotamente não devem apresentar resultados piores do que as realizadas localmente" (MACHADO; MAIA, 2017, p. 238).

Conforme você observou em seus estudos sobre a característica de transparência dos sistemas distribuídos, uma das questões que fortalecem esta característica é que, no caso de ocorrer algum erro enquanto o usuário estiver utilizando o sistema, este não deverá ser percebido. Caberá ao sistema contornar o acontecido. Esta capacidade do sistema contornar eventuais erros e fazer com que passem despercebidos para o usuário é chamada de **tratamento ou tolerância a falhas**.

Assim, não é exagero afirmar que (e você acabou de verificar isso no Quadro 2), para que o sistema seja transparente é preciso implementar adequadamente o tratamento a falhas, tanto para o contexto de hardware quanto de software.

- **Tratamento de falhas de hardware:** em sistemas distribuídos, o tratamento de falhas de hardware está relacionado à redundância, como, por exemplo, a existência de várias fontes e processadores, memórias com detecção e correção de erros, redundância dos meios de conexão, entre outros.
- **Tratamento de falhas de software:** mais complexa de se implementar, deve garantir que, no caso de uma falha no SO, por exemplo, a aplicação continue a ser executada normalmente, sem que o usuário perceba.



A diferença entre o tratamento de falhas de software em um SO de rede e em um sistema distribuído é que "enquanto em uma rede de computadores o usuário deverá se reconectar a um outro sistema em funcionamento e reiniciar sua tarefa, em um sistema distribuído o problema deve ser resolvido de forma transparente, mantendo a integridade e consistência dos dados" (MACHADO; MAIA, 2017, p. 238).

Ao implementar adequadamente o tratamento de falhas, o sistema garante que não haverá interrupções nas aplicações que estão sendo processadas pelo usuário no caso de problemas em algum componente. Desta forma, sistemas distribuídos são capazes de oferecer alta disponibilidade e confiabilidade.

Com a tolerância a falhas, é possível, também, oferecer alta disponibilidade e confiabilidade. Como existem sistemas autônomos, em caso de falha de um dos componentes, um outro sistema poderá assumir suas funções, sem a interrupção do processamento. Como as aplicações estão distribuídas por diversos sistemas, caso ocorra algum problema com um dos componentes é possível que um deles assuma, de forma transparente, o papel do sistema defeituoso (MACHADO; MAIA, 2017, p. 238).

A partir do que você estudou até agora, pode perceber que a implementação do tratamento de falhas em um sistema distribuído não é uma tarefa simples, isso porque enquanto alguns componentes falham, outros continuam em funcionamento sem que se possa tomar conhecimento destas falhas. Para ser devidamente implementado, há técnicas para tratamento de falhas, demonstradas no Quadro 3 (COULOURIS *et al.*, 2013):

QUADRO 3 – TÉCNICAS DE TRATAMENTO DE FALHAS

Técnica	Descrição
Detecção de falhas	Algumas falhas podem ser detectadas, outras não. Por exemplo, dados corrompidos em uma mensagem podem ser mais facilmente detectados do que um servidor remoto danificado. Esta técnica gerencia a ocorrência de falhas que não podem ser detectadas, mas que podem ser suspeitas.
Mascaramento de falhas	Algumas falhas detectadas podem ser ocultas ou se tornar menos sérias.
Tolerância a falhas	Como a maioria dos serviços na internet apresenta falhas, ao invés de tentar detectar e mascarar a todas elas, pode ser mais interessante projetar um sistema capaz de tolerar falhas.
Recuperação de falhas	A recuperação envolve projetar softwares de modo que o estado dos dados permanentes possa ser recuperado ou "retrocedido" após a falha de um servidor.
Redundância	Os serviços podem se tornar tolerantes a falhas com o uso de componentes redundantes

FONTE: Adaptado de Coulouris *et al.* (2013, p. 21-22)

4 EXEMPLOS DE SISTEMAS DISTRIBUÍDOS

Nesta seção, mostraremos alguns exemplos de sistemas distribuídos atualmente disponíveis. Você perceberá, a partir dos exemplos apresentados, que os sistemas distribuídos estão presentes em algumas das tecnologias mais significativas e usuais em nosso cotidiano, agregando ainda mais valor ao conhecimento sobre este tipo de sistema (COULOURIS *et al.*, 2013).

- **Pesquisa na web:** há registros de que as pesquisas realizadas na web ultrapassam os 10 bilhões por mês. Basicamente, uma pesquisa na web deve indexar todo o conteúdo disponível (estamos falando de 63 bilhões de páginas e um trilhão de endereços) em uma grande variedade, desde páginas web até arquivos de multimídia. O principal mecanismo de pesquisas na web, atualmente, é o Google, que possui uma sofisticada e complexa infraestrutura de sistema distribuído capaz de suportar cada pesquisa realizada.
- **Jogos on-line:** os jogos on-line representam um dos grandes desafios no desenvolvimento de sistemas distribuídos na atualidade, uma vez que um número muito grande de jogadores interage com um mundo virtual pela internet. Com uma quantidade de usuários cada vez maior, deve-se manter um tempo de resposta rápido de modo a garantir a experiência do usuário com o jogo, além de manter a sofisticação dos universos e ambientes construídos.



O número de jogadores de jogos on-line vem aumentando. Por isso, os sistemas devem ser capazes de suportar mais de 50.000 usuários on-line simultaneamente, sendo que o número total de jogadores pode ser dez vezes maior.

- **Negócios financeiros:** o uso de sistemas distribuídos no setor financeiro já vem ocorrendo há muito tempo em consequência, basicamente, da necessidade de acesso em tempo real a uma variedade de informações, tais como tendências no mercado de ações, por exemplo. Para tanto, os sistemas distribuídos utilizados neste setor empregam uma arquitetura muito particular, baseada em eventos. Os eventos carregam informações de interesse dos usuários (tais como o exemplo do mercado de ações mencionado anteriormente) e devem ser distribuídos de forma confiável e oportunamente.

Além dos exemplos de uso dos sistemas distribuídos, vale ainda mencionar que o constante avanço tecnológico vem marcando as tendências para as aplicações distribuídas, tais como (COULOURIS *et al.*, 2013, p. 8):

- O surgimento da tecnologia de redes pervasivas.
- O surgimento da computação ubíqua, combinado ao desejo de suportar mobilidade do usuário em sistemas distribuídos.
- A crescente demanda por serviços multimídia.
- A visão dos sistemas distribuídos como um serviço público.

RESUMO DO TÓPICO 1

Neste tópico, você aprendeu que:

- Os estudos sobre as redes de computadores e os sistemas distribuídos passaram a ter maior destaque a partir dos anos 1980.
- As redes de computadores se caracterizam pela existência de dois ou mais sistemas computacionais distintos interligados, capazes de trocar informações.
- A existência de vários *hosts* (sistemas computacionais independentes) em uma rede de computadores a torna um ambiente multiprocessado, exigindo um tipo específico de SO fracamente acoplado: os sistemas distribuídos.
- “Um sistema distribuído é um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente” (TANENBAUM; STEEN, 2007, p. 1).
- Do ponto de vista do usuário é como se houvesse um único sistema; do ponto de vista do hardware existem vários sistemas computacionais distintos, cada um com seus recursos.
- Sistemas distribuídos são chamados middleware por representarem uma camada intermediária entre o SO existente em cada computador e as aplicações utilizadas pelos usuários.
- Dentre os benefícios de se utilizar sistemas distribuídos estão: melhoria da capacidade de processamento, melhoria da capacidade de armazenamento e o compartilhamento de recursos.
- Dentre as características desejáveis em um sistema distribuído estão: escalabilidade, concorrência entre os componentes do sistema, segurança, tratamento de falhas e transparência.



- 1 A popularização e avanço dos computadores pessoais e estações de trabalho juntamente com as tecnologias de informação e comunicação, especialmente as associadas às redes de computadores, propiciou o surgimento de um novo modelo de computação, chamado modelo de rede de computadores. Esse novo modelo introduziu um novo tipo de sistema computacional, os *hosts*, e, consequentemente, a necessidade de SO fracamente acoplados. Dessa forma, indique a sentença que traz os dois tipos desses SO:
 - a) () SO de rede e os sistemas distribuídos.
 - b) () SO de rede e os *clusters*.
 - c) () Sistemas distribuídos e os *clusters*.
 - d) () SO de rede e os sistemas fracamente acoplados.
- 2 Sistemas distribuídos compreendem uma série de elementos, conceitos e práticas de projeto para a sua construção, passando por infraestrutura física e lógica de redes de computadores, e outros dispositivos com conectividade, SO, arquiteturas de sistemas, entre outros. Pode-se então formalizar definições para esse tipo de sistema que compreendam todos esses elementos. Com isso em mente, assinale a sentença que não traz uma definição correta para o conceito de sistemas distribuídos.
 - a) () Para a percepção do usuário final, um sistema distribuído é utilizado transparentemente, como se não existisse uma infraestrutura de rede de computadores dando suporte a ele, e que apesar de distribuído por essa rede, o usuário o percebe como um único sistema.
 - b) () Tendo como ponto de partida uma visão de implementação, com suporte a ambientes heterogêneos e oferecendo uma visão única de sistema, esses sistemas muitas vezes são organizados como uma camada de software chamada *middleware*.
 - c) () Para a percepção do usuário final, um sistema distribuído é utilizado transparentemente, como se não existisse uma infraestrutura de rede de computadores dando suporte a ele, e que apesar de distribuído por essa rede, o usuário o percebe como um único sistema. Essa característica de transparência é muitas vezes denominada *middleware*.
 - d) () Em uma percepção de hardware um sistema distribuído congrega uma miríade de sistemas computacionais distintos, com memórias e processadores como seus recursos. Mesmo assim, há a impressão de que existe uma única máquina uma vez que se tem um único SO gerenciando tais recursos.

3 A escalabilidade é uma característica imprescindível dos sistemas distribuídos, uma vez que ela deve manter a eficiência do sistema mesmo com a inserção de novos recursos ou usuários. Logo, na construção de um sistema distribuído existem desafios significativos para que essa sua característica seja alcançada. Com isso em mente, avalie as sentenças relacionadas a esses desafios classificando V para as sentenças verdadeiras e F para as falsas.

- () Os controles relacionados ao custo dos recursos físicos, considerando a inclusão de novos recursos, devem ser aceitáveis.
- () No controle associado à perda de desempenho, considerando a escalabilidade, a perda deve ser menor que a função do tempo de acesso aos dados do sistema.
- () Os controles relacionados ao custo dos recursos físicos, considerando a inclusão de novos recursos, estão sempre disponíveis já que se deve garantir a característica da escalabilidade.
- () No controle associado à perda de desempenho, considerando a escalabilidade, a perda não deve ser menor que a função do tempo de acesso aos dados do sistema.

Assinale a alternativa CORRETA:

- a) () V – V – F – F.
- b) () F – V – V – F.
- c) () F – F – V – V.
- d) () V – V – F – F.

4 Sistemas distribuídos compreendem uma série de características (heterogeneidade, escalabilidade, concorrência, segurança, entre outras) que devem ser alcançadas para viabilizar um projeto para um sistema dessa natureza. Como esses sistemas lidam, por vias de regra, com infraestruturas física e lógica de redes de computadores (e outros dispositivos com conectividade), SO, arquiteturas de sistemas, entre outros, as características de concorrência aos recursos e, consequentemente, sua segurança são condições necessárias a se alcançar. Com isso em mente, avalie a sentença de afirmação com a sua sentença de explicação relacionadas a essas duas características dos sistemas distribuídos.

I- A concorrência nesse tipo de ambiente é perfeitamente natural, pois existe uma chance potencialmente alta de que inúmeros usuários concorram para a utilização de um dado recurso compartilhado no sistema. Logo, esses inúmeros usuários distintos, cada um em seu dispositivo, podem compartilhar arquivos (como fotos, áudios, vídeos, entre outros) sempre que necessário. Para tanto, eles devem, de alguma maneira, possuir uma identificação no sistema (rede de origem, login, senha, protocolo de segurança, entre outros).

PORQUE

II-Parte, ou grande parte, da característica de concorrência está associada ao quesito segurança, uma vez que o sistema deve garantir a ocultação dos conteúdos das mensagens, bem como deve garantir, de maneira confiável, as identidades do emissor da mensagem e do seu receptor.

Agora, assinale a alternativa que apresenta a resposta CORRETA:

- a) () As sentenças I e II representam proposições verdadeiras, mas a II não é uma justificativa correta da I.
- b) () As sentenças I e II representam proposições verdadeiras, e a II é uma justificativa correta da I.
- c) () As sentenças I e II não representam proposições verdadeiras.
- d) () A sentença I é uma proposição verdadeira, porém a II é uma proposição falsa.

5 A escalabilidade é uma característica imprescindível dos sistemas distribuídos, uma vez que ela deve manter a eficiência do sistema mesmo com a inserção de novos recursos ou usuários. Logo, na construção de um sistema distribuído existem desafios significativos para que essa sua característica seja alcançada. Com isso em mente, avalie as sentenças relacionadas a esses desafios classificando com V as sentenças verdadeiras e F as falsas.

- () No controle associado à perda de desempenho, considerando a escalabilidade, a perda pode ser menor, ou maior, que a função do tempo de acesso aos dados do sistema. Isso dependerá do grau de escalabilidade que se quer alcançar com o sistema.
- () O controle relacionado ao custo dos recursos físicos, considerando a inclusão de novos recursos, possui sua disponibilidade diretamente proporcional ao grau de escalabilidade que se deseja dar a um determinado módulo do sistema.
- () Deve-se evitar o esgotamento dos recursos de software, porém fazer uma previsão antecipada da escalabilidade de um sistema distribuído não é uma tarefa fácil. Logo, uma das abordagens para evitar este esgotamento pode ser através da adaptação às mudanças.
- () Deve-se evitar o esgotamento dos recursos de software, para isso é imprescindível uma previsão antecipada da escalabilidade, mesmo que para um sistema distribuído isso não seja uma tarefa fácil. De qualquer forma, a adaptação às mudanças não pode ser uma abordagem sugerida uma vez que não se trata de uma medida eficaz.

Agora, assinale a alternativa CORRETA:

- a) () V – V – F – F.
- b) () F – V – V – F.
- c) () F – F – V – F.
- d) () V – V – F – F.

ARQUITETURA PARA SISTEMAS DISTRIBUÍDOS

1 INTRODUÇÃO

Como aprendemos na Unidade 1, os sistemas distribuídos são sistemas complexos de serem construídos dada a natureza de suas atividades fim, bem como o tipo de ambiente heterogêneo no qual eles as executam. Sendo assim, apresentar formas que consigam descrever e modelar a sua estrutura é uma condição fundamental para a construção desse tipo de sistema. Neste contexto, temos que possuir a capacidade de descrever a estrutura de um sistema distribuído na forma de seus modelos de arquitetura.

Desta forma, assim como os computadores que possuem seus componentes físicos (hardware) e lógicos (software), os sistemas distribuídos são representados por elementos físicos (mais óbvios e explícitos, pois são visuais) e por seus elementos lógicos. Os elementos lógicos são representados, por exemplo, pela aplicação de um sistema distribuído, suas tarefas e seus aspectos de comunicação e segurança, bem mais expressivos e significativos do que os seus irmãos físicos. Vejamos os modelos de arquitetura associados à existência de um sistema distribuído.



Acadêmico, sugerimos relembrar o conceito de arquitetura apresentado no Tópico 3 da Unidade 1.

Agora, iniciemos os nossos estudos aprendendo sobre as arquiteturas físicas e as arquiteturas para sistemas distribuídos.

2 ARQUITETURAS FÍSICAS

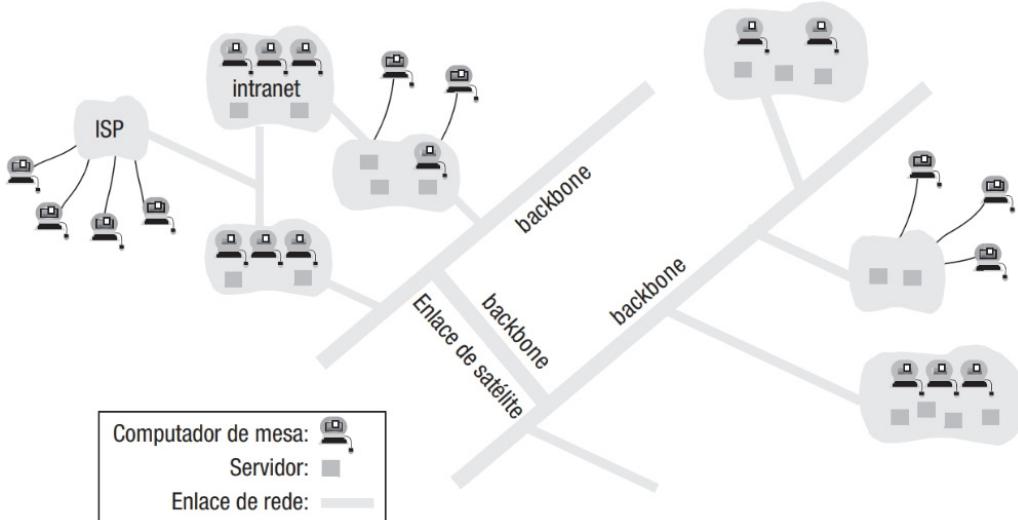
Como já comentado anteriormente, esses modelos apresentam e tratam da maneira mais explícita (direta, visual) a apresentação de um sistema distribuído, pois lidam com a composição de hardware desse tipo de sistema, focando no contexto dos dispositivos interconectados (incluindo os móveis) e suas redes de interconexão com toda a infraestrutura necessária (COULOURIS *et al.*, 2013).

Apesar de esses modelos tratar dos atributos físicos dos sistemas distribuídos, eles devem ser abstratos o suficiente para que se possa manter a comunicação entre os dispositivos interligados em rede, e suas ações coordenadas através da passagem de mensagens, independentes uns dos outros (COULOURIS *et al.*, 2013).

Vejamos agora as gerações de sistemas distribuídos identificadas de acordo com as tecnologias existentes em cada período (COULOURIS *et al.*, 2013):

- 1- **Sistemas distribuídos primitivos:** situados na passagem da década de 1970 para a década de 1980, surgiram como uma resposta de infraestrutura para as redes locais, que permitem o compartilhamento de serviços de impressão, serviços de arquivos e transferências de e-mail e arquivos pela internet. Sua quantidade de nós era limitada a um número entre 10 e 100 (rede local) e possuíam uma velocidade de internet muito limitada. Além disso, rodavam em um ambiente homogêneo com padrões de qualidade de serviço muito baixos.
- 2- **Sistemas distribuídos adaptados para internet:** a internet foi a grande impulsionadora dessa geração através de seu surgimento na década de 1990 e seu amadurecimento com tecnologias como o mecanismo de busca Google nascido em 1996. Nesse período, as redes tomam uma abrangência global através da internet (Figura 5) adquirindo uma capacidade de comunicação não apenas organizacional, mas global, com um significativo crescimento do seu número de nós. Com isso, cresce exponencialmente o nível de heterogeneidade dos ambientes desses sistemas considerando as redes, as arquiteturas de computador, os tipos de sistemas operacionais, e as linguagens empregadas para a criação de sistemas e as equipes de desenvolvimento envolvidas.

FIGURA 5 – CONFIGURAÇÃO TÍPICA DE NÓS NA INTERNET



FONTE: Couloris *et al.* (2013, p. 9)

- 3- Sistemas distribuídos contemporâneos:** aqui, um outro elemento é adicionado para aumentar a complexidade dos sistemas distribuídos, em contrapartida aos nós estáticos representados pelos computadores de mesa e notebooks: tratam-se dos dispositivos móveis. Eles advêm como uma das tendências apontadas por Coulouris *et al.* (2013, p. 8): “o nascimento das redes pervasivas e da computação ubíqua, o aumento de demanda por serviços multimídia e a incorporação dos sistemas distribuídos como uma forma de serviço público”. Vale ressaltar que o Tópico 5, desta unidade, trata exatamente deste tema. Isso traz como reflexo, ainda maior, o aumento da heterogeneidade dos ambientes, a interconexão de centenas a milhares de nós explorando elementos computacionais complexos encontrados na computação em grade (grid) que você estudará mais adiante.
- 4- Sistemas distribuídos de sistemas:** ambiente mais complexo formado por uma série de subsistemas, que representam, eles próprios, sistemas que são combinados para executar tarefas especiais, por exemplo, um sistema ambiental para previsão de enchentes. Nele, redes de sensores monitoram o estado de parâmetros ambientais relacionados a rios, terrenos propensos à inundação, efeitos das marés, entre outros. Então, através de simulações (sistema de previsão), pode-se prever a ocorrência de enchentes ou outros desastres ambientais. O Quadro 4 apresenta as gerações dos sistemas distribuídos.

QUADRO 4 – GERAÇÕES DOS SISTEMAS DISTRIBUÍDOS

Sistemas distribuídos	Primitivos	Adaptados para Internet	Contemporâneos
<i>Escala</i>	Pequenos	Grandes	Ultragrandes
<i>Heterogeneidade</i>	Limitada (normalmente, configurações relativamente homogêneas)	Significativa em termos de plataformas, linguagens e <i>middleware</i>	Maiores dimensões introduzidas, incluindo estilos de arquitetura radicalmente diferentes
<i>Sistemas abertos</i>	Não é prioridade	Prioridade significativa, com introdução de diversos padrões	Grande desafio para a pesquisa, com os padrões existentes ainda incapazes de abranger sistemas complexos
<i>Qualidade de serviço</i>	Em seu início	Prioridade significativa, com introdução de vários serviços	Grande desafio para a pesquisa, com os serviços existentes ainda incapazes de abranger sistemas complexos

FONTE: Coulouris *et al.* (2013, p. 41)

3 ARQUITETURAS PARA SISTEMAS DISTRIBUÍDOS

Como você percebeu, as arquiteturas físicas tratam, de uma maneira geral, dos elementos físicos existentes em um ambiente de um sistema distribuído. Já as arquiteturas para sistemas distribuídos descrevem esses sistemas do ponto de vista de suas tarefas computacionais e, também, da relação de comunicação entre os seus elementos computacionais (computadores individuais e seus agregados — *clusteres*) suportados pelas interconexões das redes que os conectam. Logo, a arquitetura de um sistema estrutura os seus componentes especificados

separadamente, e suas inter-relações, com o objetivo de garantir que essa arquitetura atenda as atuais e as futuras cargas de solicitações impostas sobre o sistema. A arquitetura tem um impacto direto nas qualidades de confiabilidade, adaptabilidade, rentabilidade e condição gerenciável do sistema.

Neste tópico, vamos descrever os principais modelos de arquitetura empregados nos sistemas distribuídos — seus estilos arquitetônicos — que na apresentação de Coulouris *et al.* (2013) consideram que a construção de uma arquitetura para sistemas distribuídos pode ser visualizada de três formas: elementos arquitetônicos, padrões arquitetônicos e soluções de *middleware* associadas.

3.1 ELEMENTOS ARQUITETÔNICOS

Esses elementos envolvem as entidades (dispositivos) que estabelecem uma comunicação e como essa comunicação ocorre, para a realização de alguma função, funções e responsabilidades. E, finalmente, como essas entidades estão mapeadas na infraestrutura fisicamente distribuída do sistema.

- **Entidades:** sensores (em uma rede de sensores), *threads*, objetos, componentes, serviços web, entre outros.
- **Comunicação:** comunicação entre processos, invocação remota e comunicação indireta.
- **Funções e responsabilidades:** as entidades interagem (se comunicam) para, por exemplo, suportar uma sessão de bate-papo em uma rede social.

O Quadro 5 apresenta as entidades que podem se comunicar em um ambiente distribuído e a forma como elas podem realizar essa comunicação.

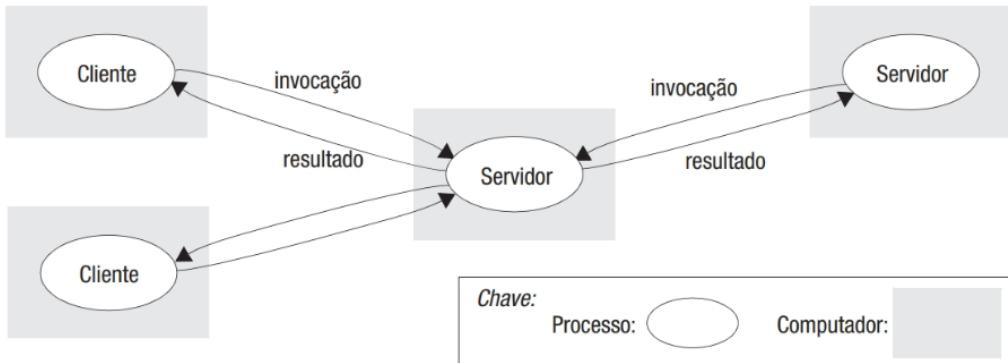
QUADRO 5 – ENTIDADES E PARADIGMAS DE COMUNICAÇÃO

Entidades em comunicação (o que se comunica)		Paradigmas de comunicação (como se comunicam)		
Orientados a sistemas	Orientados a problemas	Entre processos	Invocação remota	Comunicação indireta
Nós	Objetos	Passagem de mensagem	Requisição-resposta	Comunicação em grupo
		Soquetes	RPC	Publicar-assinar
		Multicast	RMI	Fila de mensagem Espaço de tupla DSM
Processos	Componentes			
	Serviços Web			

FONTE: Coulouris *et al.* (2013, p. 46)

A Figura 6 apresenta um modelo (*peer-to-peer*) no qual as funções e a responsabilidade de cada par (*peer* - cliente/servidor) são distribuídas de maneira equivalente entre a infraestrutura física do sistema.

FIGURA 6 – EXEMPLO DA ARQUITETURA EM PARES (PEERS)



FONTE: Coulouris et al. (2013, p. 47)

- **Mapeamento físico:** deve considerar a forma de comunicação entre as entidades, a confiabilidade de determinados dispositivos (especialmente os servidores), sua carga atual, a qualidade da comunicação entre os diferentes dispositivos, entre outros. Algumas estratégias de mapeamento são: serviços em vários servidores, uso de cache, código móvel e agentes móveis.

3.2 PADRÕES ARQUITETÔNICOS

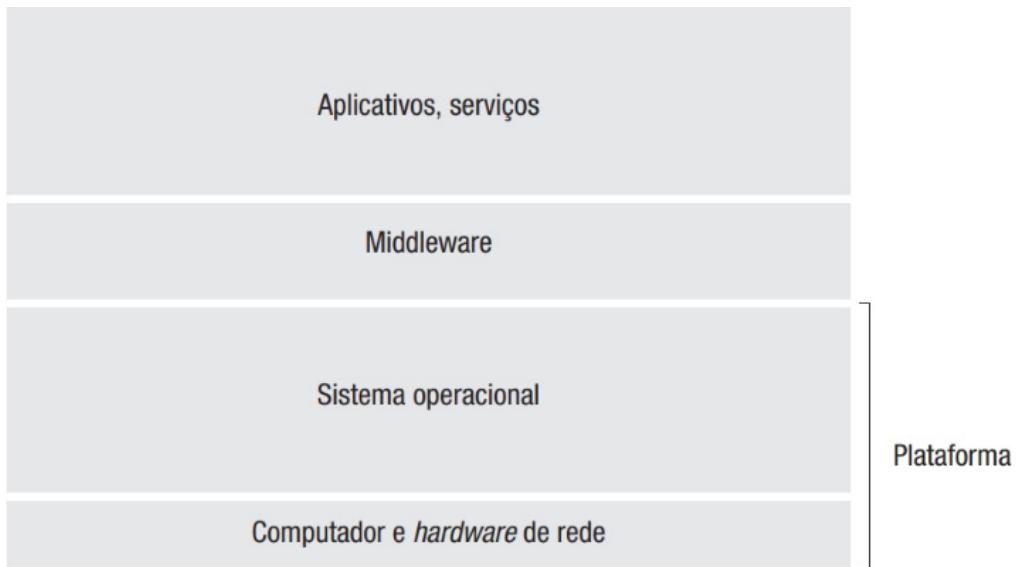
Tratam-se de padrões recorrentes já experimentados em várias situações que podem ser usados na solução de problemas diferentes. Aqui você tem a oportunidade de conhecer as arquiteturas de camadas lógicas (*layer*), as de camadas físicas (*tier*), além do conceito dos chamados clientes leves ou “magros” (*thin clients*).



Padrão é algo usado como exemplo especialmente para copiar qualquer arranjo, combinação, regularmente repetido. Maneira particular com a qual algo é realizado, é organizado ou acontece (PATTERN, 2018).

- **Camadas lógicas:** nessa abordagem o sistema é dividido em camadas, nas quais cada uma utiliza as funções oferecidas pela camada lógica inferior. A Figura 7 apresenta uma abstração lógica de uma estrutura em camadas para a construção de sistemas distribuídos que permite a organização de seus aplicativos e serviços.

FIGURA 7 – CAMADAS LÓGICAS DE SERVIÇOS DE SOFTWARE E HARDWARE EM SISTEMAS DISTRIBUÍDOS

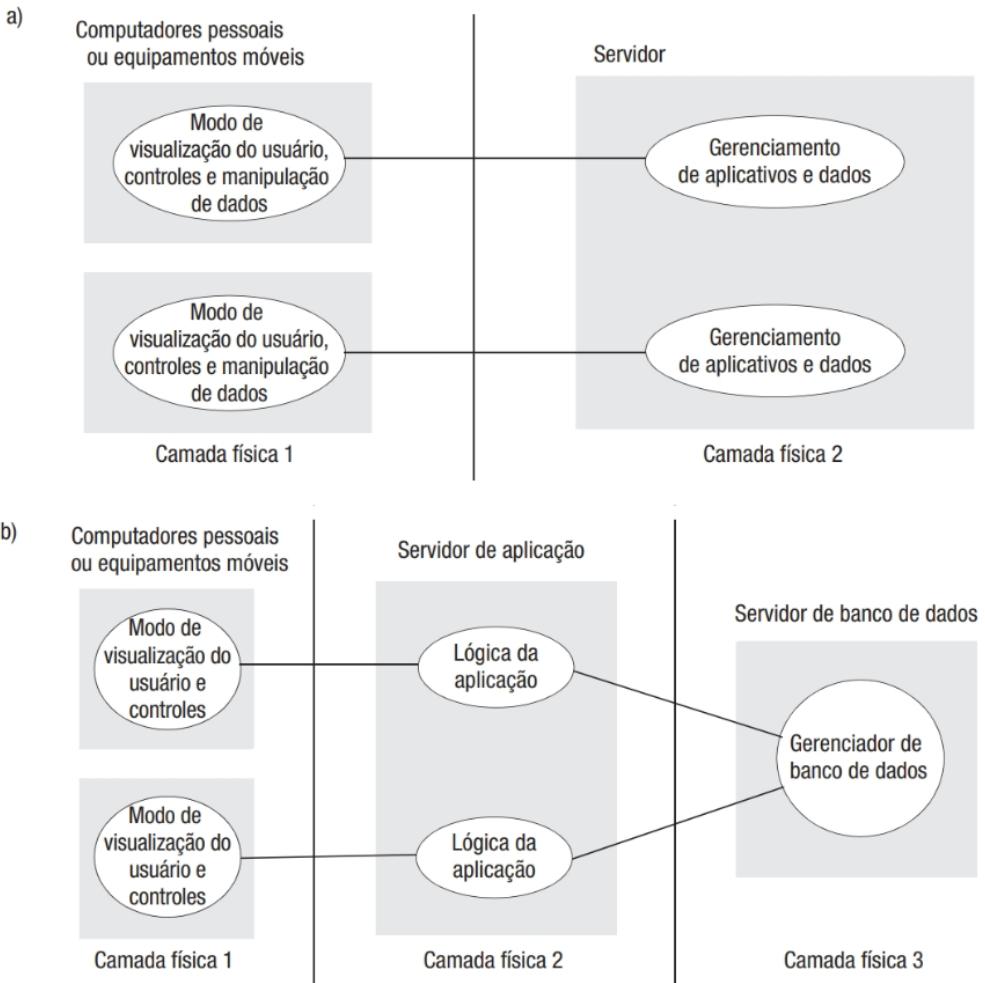


FONTE: Coulouris et al. (2013, p. 52)

- **Arquitetura de camadas físicas:** complementam suas irmãs, as camadas lógicas. As camadas físicas organizam as funcionalidades de uma determinada camada lógica, além de ser responsável por essas funcionalidades nos servidores apropriados.

Lembre-se que as camadas lógicas tratam de uma organização vertical desses serviços (com suas funcionalidades ou operações) em camadas de abstração para simplificar os seus acessos a recursos. A Figura 8 apresenta a decomposição funcional de um dado serviço ou aplicação entre as camadas físicas de um sistema distribuído. Perceba que as camadas são representadas pelos dispositivos pessoais dos usuários, por um servidor de banco de dados, e no caso do modelo, em três camadas há a inclusão de um servidor de aplicação responsável por armazenar as entidades com as operações e funcionalidades do sistema, relegando a camada de dispositivos apenas as chamadas visuais a essas funcionalidades e a camada de banco de dados o gerenciamento dos dados manipulados por elas.

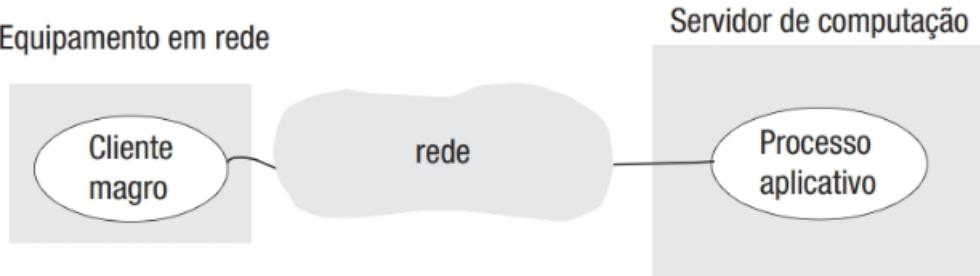
FIGURA 8 – ARQUITETURA DE DUAS E DE TRÊS CAMADAS FÍSICAS



FONTE: Coulouris et al. (2013, p. 54)

- **Cientes “magros” (*thin clients*):** com a tendência da computação distribuída em otimizar o processamento dos dispositivos clientes, além de diminuir a quantidade de códigos fontes nos mesmos adicionando-os aos serviços da internet, principalmente com o amadurecimento e popularização da computação em nuvem, foi reacendido o interesse da indústria de software nos chamados *thin clients*, isto é, os clientes magros. Esse conceito de dispositivo baseia-se na existência de uma camada de software que dá suporte a uma interface constituída por janelas (local para o usuário), executando operações ou também acessando serviços em um servidor remoto, como mostra a Figura 9.

FIGURA 9 – CLIENTES MAGROS E SERVIDORES



FONTE: Couloris et al. (2013, p. 57)

3.3 SOLUÇÕES DE MIDDLEWARE ASSOCIADAS

Um *middleware* representa uma abstração em nível mais alto de programação no desenvolvimento de sistemas distribuídos. Fazendo uso dos elementos e padrões arquitetônicos, que você viu anteriormente, através de camadas lógicas, essas soluções são capazes de simplificar a heterogeneidade da infraestrutura física, e até lógica, existente e promover a interoperabilidade e a portabilidade entre as entidades de um sistema distribuído.

Algumas tecnologias para o desenvolvimento de soluções com foco em *middleware* são: RM-ODP, CORBA, Java RMI, Fractal, OpenCOM, JBoss, Scribe JMS, Websphere MQ, Apache Axis, The Globus Toolkit, Pastry, Tapestry, Squirrel, OceanStore, Ivy e Gnutella.

4 ARQUITETURAS FUNDAMENTAIS

Os modelos fundamentais utilizam três visões para avaliar os sistemas distribuídos de maneira individual: (1) os modelos de interação, que tratam diretamente da comunicação entre os elementos do sistema; (2) os modelos de falha, que lidam com condições adversas que podem conduzir o sistema a funcionar incorretamente ou até mesmo parar de funcionar; e (3) os modelos de segurança, que assumem que o funcionamento de um sistema distribuído passa pela proteção contra tentativas de interferência em seus elementos ou de roubo dos dados manipulados por ele. Esses modelos consideram que um sistema distribuído é composto por processos comunicantes de mensageria através de uma infraestrutura de redes de dispositivos, e são criados para conter os elementos (componentes) primordiais que permitem entender, e raciocinar, sobre comportamentos específicos de um sistema.



Mensageria é um método de comunicação entre componentes de software ou aplicações. Um sistema de mensageria é um facilitador par a par (*peer-to-peer*): um cliente pode enviar mensagens para e receber mensagens de qualquer outro cliente. Cada cliente conecta-se a um agente que provê recursos para enviar, receber e ler mensagens (THE JAVA EE 6, 2010).

Os modelos fundamentais têm como objetivos:

- 1- Evidenciar todas as suposições relevantes sobre o sistema modelado.
- 2- Generalizar o que é possível ou impossível, considerando essas suposições.

Você percebeu que no primeiro objetivo as suposições giram em torno de aspectos de interação, falhas e segurança, e que no contexto do segundo essas suposições são exploradas e detalhadas? Que tal agora explorarmos essas suposições?

- **Modelo de interação:** é estabelecido por processos que trocam mensagens, gerando uma comunicação (fluxo de informações), que é então coordenada (sincronizada e ordenada com suas operações). No âmbito dos sistemas distribuídos essas interações ganham um papel de destaque, uma vez que o modelo de interação deve refletir o fato de que a comunicação é sujeita a atrasos com frequente, e não desprezível, duração. Logo, a exatidão com que esses processos independentes precisam ser coordenados, limita-se aos atrasos de sua comunicação e pela complexidade da manutenção da mesma “noção” de tempo entre todos os dispositivos envolvidos nessas interações. Lembre-se que em um ambiente distribuído seus componentes estão espalhados pela rede, inclusive remotamente. Alguns exemplos dessas interações podem ser observados:
 - Na cooperação entre os processos dos serviços de DNS (*Domain Name Service* – serviço para nomeação de domínios de computadores na internet) que dividem e replicam seus dados em diferentes servidores através da internet.
 - Um conjunto de processos *peer-to-peer* quando, por exemplo, você está baixando algum arquivo multimídia da categoria Torrent.
- **Modelo de falhas:** as coordenações compartilhadas, muitas vezes remotamente, nos ambientes de sistemas distribuídos tornam crítica a ocorrência de uma falha em qualquer um dos dispositivos envolvidos em uma comunicação, e consequentemente, o tratamento dessas possíveis falhas (incluindo falhas de software) na infraestrutura física ou de rede que interliga as entidades distribuídas. Logo, um modelo de falhas define e classifica as falhas em um ambiente distribuído e com ele torna-se possível o tratamento de possíveis indisponibilidades do sistema, pois esse modelo fornece uma base para a análise dos efeitos potenciais dessas falhas auxiliando no projeto de sistemas capazes de tolerar exceções e de continuar funcionando corretamente. Em um sistema distribuído, tanto os processos como os canais de comunicação

podem falhar. Hadzilacos e Toueg (1994 *apud* COULOURIS *et al.*, 2013, p. 68) “fornecem uma taxonomia que diferencia as falhas de processos e as falhas nos canais de comunicação, sendo elas: falhas por omissão, falhas arbitrárias e falhas de sincronização”.

- Falhas por omissão: representam as situações nas quais um processo ou canal de comunicação não executam as ações que deveriam.
- Falhas arbitrárias: representam o pior cenário entre os grupos de falhas, pois não executam arbitrariamente passos desejados de um processo ou mesmo efetuam um dado processamento de forma indesejada. Dessa forma, um processo pode gerar dados com valores incorretos ou mesmo retornar um valor incorreto a uma chamada realizada por outro processo. Logo, o agravante desse grupo de falhas é não poderem ser detectadas, verificando-se o processo responde corretamente as suas chamadas, pois ele poderia omitir arbitrariamente a resposta.
- Falhas por sincronização (temporização): estão relacionadas às execuções de sistemas distribuídos síncronos. Nesse tipo de ambiente, no qual limites de tempo são definidos para a execução de seus processos e o tempo de entrega de mensagens, uma falha pode tornar indisponíveis as respostas para os clientes dentro de um intervalo de tempo prefixado. Aplicações multimídia, com seus canais de áudio e vídeo, sofrem particular influência dessas falhas, uma vez que, manipulações de vídeo podem gerar transferências de grandes volumes de dados. Sendo assim, torna-se necessário distribuir as informações geradas sem falhas de temporização. Impõe, dessa forma, condições muito especiais sobre o sistema operacional e sobre o sistema de comunicação.
- **Modelo de segurança:** uma das características mais significativas dos sistemas distribuídos (filosofia de sistemas abertos) os deixam expostos a ataques externos e internos. Um modelo de segurança identifica as formas desses ataques, classificando-as, e gerando uma base de potenciais ameaças ao sistema, fazendo com que a sua resistência aos ataques se torne mais eficiente e eficaz, isto é, mais preventiva.



Veremos mais detalhes sobre consistência e tolerância às falhas no Tópico 2 da Unidade 3, e sobre segurança no Tópico 3 da mesma unidade.

RESUMO DO TÓPICO 2

Neste tópico, você aprendeu que:

- Dada a natureza complexa dos sistemas distribuídos, é importante o conhecimento de seus modelos de arquitetura, levando-se em conta os elementos físicos e lógicos.
- As arquiteturas físicas tratam da composição de hardware dos sistemas distribuídos, incluindo dispositivos interconectados e suas redes de interconexão.
- As arquiteturas físicas evoluíram ao longo do tempo conforme também evoluía a tecnologia disponível, organizando os sistemas distribuídos em três gerações: sistemas distribuídos primitivos, sistemas distribuídos adaptados para internet, sistemas distribuídos contemporâneos e sistemas distribuídos de sistemas.
- As arquiteturas para sistemas distribuídos os descrevem do ponto de vista das tarefas computacionais que são executadas por eles, tendo impacto direto sobre aspectos como confiabilidade, adaptabilidade, rentabilidade e condições de gerenciamento do sistema.
- Os elementos arquitetônicos consideram: (1) entidades (por exemplo, sensores, *threads*, serviços *web*, entre outros); (2) comunicação (entre processos, por exemplo); (3) funções e responsabilidades (suporte à comunicação entre as entidades, por exemplo); e (4) mapeamento físico (forma de comunicação entre as entidades).
- Os padrões arquitetônicos envolvem as arquiteturas de camadas lógicas (*layer*), camadas físicas (*tier*) e clientes magros (*thin clients*).
- Na arquitetura de camadas lógicas o sistema é dividido em camadas, em que cada uma utiliza as funções oferecidas pela camada inferior.
- A arquitetura de camadas físicas complementa a arquitetura de camadas lógicas, organizando as funcionalidades de uma determinada camada lógica nos servidores apropriados.
- Na arquitetura *thin client* há uma camada de software que dá suporte a uma interface local para o usuário, executando operações ou também acessando serviços em um servidor remoto.
- Um *middleware* está associado às camadas lógicas de um sistema distribuído e é tratado como uma solução capaz de promover a interoperabilidade e a portabilidade entre as entidades do sistema.
- São três as visões de arquiteturas fundamentais: (1) modelos de interação (tratam da comunicação entre os elementos do sistema); (2) modelos de falha (ratam situações que poderiam levar o sistema a funcionar incorretamente ou parar de funcionar); e (3) modelos de segurança (proteção contra tentativas de interferência dos elementos ou de roubo dos dados manipulados por ele).



- 1 Os sistemas distribuídos constituem um tipo especial de sistema pois rodam em ambientes heterogêneos (plataformas 32 e 64 bits, diferentes sistemas operacionais, diferentes tipos de dispositivos, dispositivos e servidores distribuídos globalmente, entre outros) e precisam que seus componentes (físicos e lógicos) abstraiam essa miríade de ambientes para que o usuário tenha a impressão de que todos os processos, serviços, operações e tarefas acessadas estão sendo executados apenas na rede a qual ele está conectado. Dessa forma, esses sistemas são construídos de maneira que o seu hardware e o seu software consigam abstrair especificidades dos dispositivos, das tecnologias e da infraestrutura de rede empregadas.

Avalie as sentenças e a relação proposta entre elas tendo em mente o cenário apresentado anteriormente.

I- Mesmo os modelos que lidam com as características físicas dos sistemas distribuídos precisam de uma medida de abstração para garantir uma comunicação entre os seus dispositivos em rede, e suas trocas de mensagens, independentes entre si.

PORQUE

II-As arquiteturas para sistemas distribuídos apresentam tais sistemas sob uma visão de atividades computacionais e sua relação de comunicação com os componentes computacionais suportados pelas infraestruturas das redes que os conectam.

Agora, assinale a alternativa que apresenta a resposta CORRETA:

- a) () As sentenças I e II representam proposições verdadeiras, e a II é uma justificativa correta de I.
- b) () As sentenças I e II representam proposições verdadeiras, mas a II não é uma justificativa correta de I.
- c) () A sentença I representa uma proposição verdadeira, e a II uma proposição falsa.
- d) () A sentença I representa uma proposição falsa, e a II uma proposição verdadeira.

- 2 As implementações de sistemas distribuídos atuais conseguem criar abstrações significativas para os seus atributos físicos, permitindo que a comunicação entre as conexões de seus dispositivos interligados em rede coordene ações através da troca de mensagens, mantendo-os independentes entre si. Até essa condição ser alcançada, esses sistemas passaram por algumas gerações marcadas pelo tipo de tecnologias existentes aplicadas em cada uma delas. Com isso em mente, avalie as sentenças e indique a INCORRETA no tocante as gerações dos sistemas distribuídos.

- a) () A contemporaneidade dos sistemas distribuídos é alcançada quando dispositivos diferentes dos computadores pessoais e notebooks, mas com o mesmo conceito, são incluídos como elementos integrantes desses sistemas, tratam-se dos dispositivos móveis.
- b) () Inicialmente a característica “distribuída” dos sistemas distribuídos surgiu como uma resposta natural ao nascimento das redes de computadores e ao seu amadurecimento. Os serviços utilizados estavam baseados principalmente na transferência de arquivos através da rede, inclusive da internet e no compartilhamento de recursos físicos existentes nesses ambientes.
- c) () A contemporaneidade dos sistemas distribuídos é alcançada quando dispositivos diferentes dos computadores pessoais e notebooks, mas com o mesmo conceito, são incluídos como elementos integrantes desses sistemas, tratam-se dos dispositivos móveis. Nesse contexto, novos serviços são tratados por esses sistemas, como os de multimídias, e o crescimento da heterogeneidade e dos dispositivos conectados, aumenta significativamente.
- d) () Inicialmente a característica “distribuída” dos sistemas distribuídos surgiu como uma resposta natural ao nascimento das redes de computadores e ao seu amadurecimento. Os serviços distribuídos utilizados estavam baseados principalmente na transferência de arquivos através da rede, inclusive da internet, e no compartilhamento de recursos físicos existentes nesses ambientes. Apesar disso, esses primeiros sistemas já rodavam em ambiente heterogêneos, mas seguindo padrões de qualidade de serviço muito baixos.
- 3 As implementações de sistemas distribuídos atuais conseguem criar abstrações significativas para os seus atributos físicos, permitindo que a comunicação entre as conexões de seus dispositivos interligados em rede coordene ações através da troca de mensagens, mantendo-os independentes entre si. Até essa condição ser alcançada esses sistemas passaram por algumas gerações marcadas pelo tipo de tecnologias existentes aplicadas em cada uma delas. Com isso em mente, avalie as sentenças e indique qual a correta no tocante às gerações dos sistemas distribuídos:
- a) () A característica de expansibilidade dos sistemas distribuídos já era considerável (grande) mesmo na primeira geração desses sistemas.
- b) () A era da internet inaugura uma nova preocupação (característica) significativa relacionada aos sistemas distribuídos, a qualidade de serviço.
- c) () As características de expansibilidade e qualidade de serviço já nasceram maduras nos sistemas distribuídos, uma vez que do seu surgimento os ambientes computacionais já estavam maduros e as suas tecnologias.
- d) () A era da internet inaugura uma nova preocupação (característica) significativa relacionada aos sistemas distribuídos, a expansibilidade

4 Sistemas distribuídos compreendem uma série de elementos, conceitos e práticas de projeto para a sua construção. Passando por infraestrutura física e lógica de redes de computadores, e outros dispositivos com conectividade, SO, arquiteturas de sistemas, entre outros. Pode-se então formalizar definições para esse tipo de sistema que compreendam todos esses elementos. Com isso em mente, avalie as sentenças sobre as características dos sistemas distribuídos:

- () Os sistemas distribuídos precisam de um SO específico que lide com códigos distribuídos.
- () As aplicações desenvolvidas são desenvolvidas com características que as permitam se comunicar com outras aplicações, em máquinas diferentes e remotas, para a realização das atividades de um sistema distribuído.
- () As arquiteturas físicas e aquelas de software de servidores seguem estruturas e princípios que permitem aos elementos dos sistemas distribuídos se comunicarem de maneira remota para a realização das suas atividades dando ao usuário uma impressão de rede local.
- () As arquiteturas físicas e aquelas de software de servidores seguem estruturas e princípios que permitem aos elementos dos sistemas distribuídos se comunicarem de maneira remota para a realização das suas atividades dando ao usuário uma visão de comunicação distribuída entre os vários dispositivos que compõem o sistema.

Agora, assinale a alternativa CORRETA:

- a) () V – V – V – V.
- b) () V – V – V – F.
- c) () V – F – F – V.
- d) () F – F – V – V.

5 Elementos para uma arquitetura de sistemas distribuídos envolvem desde aspectos lógicos, como aspectos físicos que estão mutuamente relacionados entre si, e relacionados com a complexidade desse tipo de sistema. Dessa forma, avalie as sentenças relacionadas aos aspectos de arquitetura para os sistemas distribuídos:

- I- O suporte a uma conversa em uma determinada rede social através de um dispositivo móvel, por exemplo, diz respeito apenas ao aspecto de comunicação em um ambiente de sistemas distribuídos.
- II- O suporte a uma conversa em uma determinada rede social através de um dispositivo móvel, por exemplo, diz respeito tanto ao aspecto de responsabilidade quanto ao de comunicação em um ambiente de sistemas distribuídos.
- III- O suporte a uma conversa em uma determinada rede social através de um dispositivo móvel, por exemplo, diz respeito apenas ao aspecto de responsabilidade em um ambiente de sistemas distribuídos.
- IV- As entidades relacionadas aos aspectos de arquitetura dos sistemas distribuídos podem ser físicas (sensores e controladores, por exemplo) quanto lógicas como trechos de códigos de programas.

Agora, indique a alternativa CORRETA:

- a) () As sentenças II e III estão corretas.
- b) () As sentenças I e IV estão corretas.
- c) () As sentenças II e IV estão corretas.
- d) () As sentenças I e III estão corretas.

PROCESSOS

1 INTRODUÇÃO

Os SO representam um componente crucial de um sistema computacional, pois eles lidam com todo o aspecto de gerenciamento e escalonamento dos processos (e suas *threads*) criados pelas aplicações dos usuários. Os SO aumentam de complexidade conforme os ambientes com os quais eles lidam, como ambientes monotarefa, multitarefa, com múltiplos processadores e, assim, os ambientes dos sistemas distribuídos também tem sua complexidade aumentada. Estes últimos representam um dos mais complexos ambientes para o gerenciamento por parte dos SO, pois trabalham com plataformas heterogêneas de hardware e software, ambientes espalhados geograficamente, vários tipos de dispositivos, entre outros.

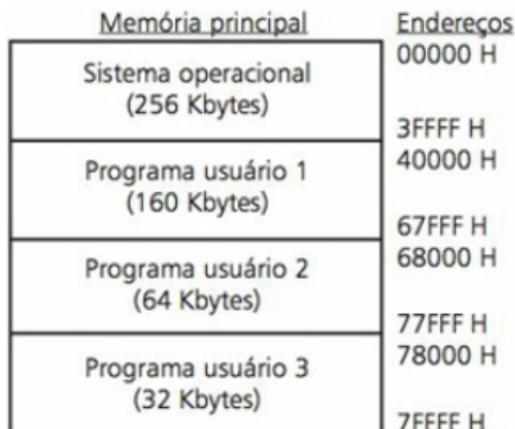
Logo, você percebeu que existe uma relação de cooperação, trabalho em equipe, e dependência significativa entre os SO e os sistemas distribuídos? Veja, você pode perceber isso nos conceitos dos conteúdos tratados nesse tópico, como processo e *threads*, que já foram abordados em nossa Unidade 1 sobre fundamentos de sistemas operacionais, e agora são retomados aqui considerando o contexto de sistemas distribuídos. Vamos, então, lembrar alguns conceitos e depois associá-los ao contexto dos sistemas distribuídos?

2 THREADS

Oliveira, Carissimi e Toscani (2010) alertam que, primeiramente, é importante distinguir entre um programa e sua execução. Vejamos um exemplo, imagine que neste momento você está acessando o sistema Gioconda, e que muitos de seus colegas podem também estar utilizando simultaneamente o sistema. Agora pense que existe apenas um sistema Gioconda e que para você e cada um de seus colegas existe um processo individual executando o sistema. Dessa forma, cada processo representa uma execução independente do sistema Gioconda para cada um de vocês.

Então, um processo pode ser definido como a execução individual de um programa. Veja que um programa é apenas uma sequência de instruções codificadas em uma linguagem de programação (algo passivo que não altera o seu próprio estado). Já o processo é um componente ativo que altera, obrigatoriamente, o seu estado (durante a execução de um programa) através de chamadas de sistema ao executar esses programas (OLIVEIRA; CARISSIMI; TOSCANI, 2010). Tomando o exemplo anterior do Gioconda, você e os seus colegas manipulam dados individuais de matrícula, disciplina, avaliações, protocolos, mensagens no AVA (Ambiente Virtual de Aprendizagem), entre outros. A Figura 10 apresenta um pequeno modelo da relação programa e processo (programa em execução).

FIGURA 10 – RELAÇÃO DE UM PROGRAMA E SEUS PROCESSOS



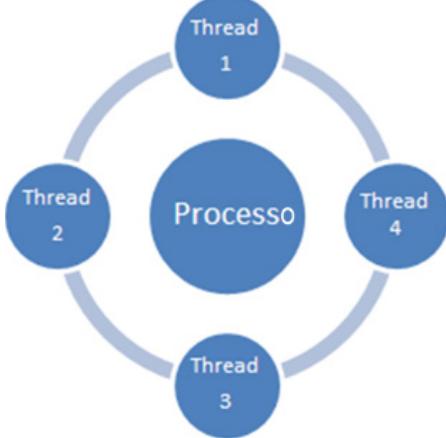
FONTE: Oliveira, Carissimi e Toscani (2010, p. 38)

Tanenbaum e Steen (2007) explicam que, para cada um dos programas em execução da Figura 10, o SO deve assegurar que todos executem sem interferência entre si. Isto é, sem que cada um dos programas afete de algum modo a corretude do comportamento individual um do outro. Isso não lembra você das funções do SO discutidas na Unidade 1?

Sabe-se então, que um processo consome recursos como espaço de endereçamento, descritores de arquivos abertos, permissões de acesso, quotas, entre outros. Em outras palavras, um processo pode consumir bastante memória e processamento. Além desses recursos, ele associa a sua existência um fluxo de execução. E para esse fluxo de execução dá-se a denominação de *thread* (componente indissociável dos processos, em termos de SO ou programação para computadores), que pertence a um único processo, mas que todo processo pode possuir mais de um (conceito de *multithreading*).

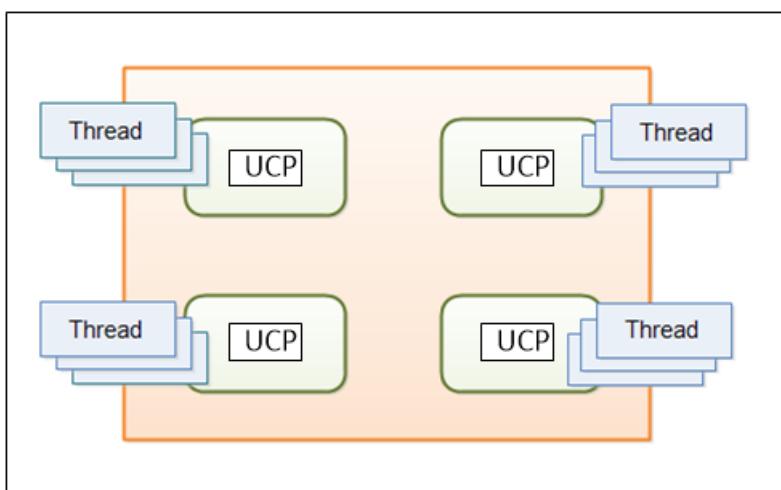
As *threads* existem no interior de um processo e compartilham entre elas os recursos desse processo, sendo consequentemente, muitas vezes, chamadas de processos leves (*Lightweight Process - LWP*) (OLIVEIRA; CARISSIMI; TOSCANI, 2010). A Figura 11 pode ser visualizada como um exemplo de uso de *thread* única (*monothread*), se a abstrairmos para apenas uma *thread*. Já a Figura 12 pode ser entendida como um exemplo de *multithreading*.

FIGURA 11 – PROCESSO E THREAD



FONTE: <https://mk0resourcesinfm536w.kinstacdn.com/wp-content/uploads/061813_1239_Multithread1.png>. Acesso em: 2 ago. 2019.

FIGURA 12 – MODELO MULTITHREADING



FONTE: <<https://hub.packtpub.com/wp-content/uploads/2018/04/image1-1.png>>. Acesso em: 2 ago. 2019.

Sobre as *threads*, Tanenbaum e Steen (2007) afirmam que elas representam uma técnica de programação importante para dividir grandes aplicações em partes menores, que podem ser executadas ao mesmo tempo e apresentam duas formas para a sua implementação: nível de usuário e nível de núcleo do sistema.

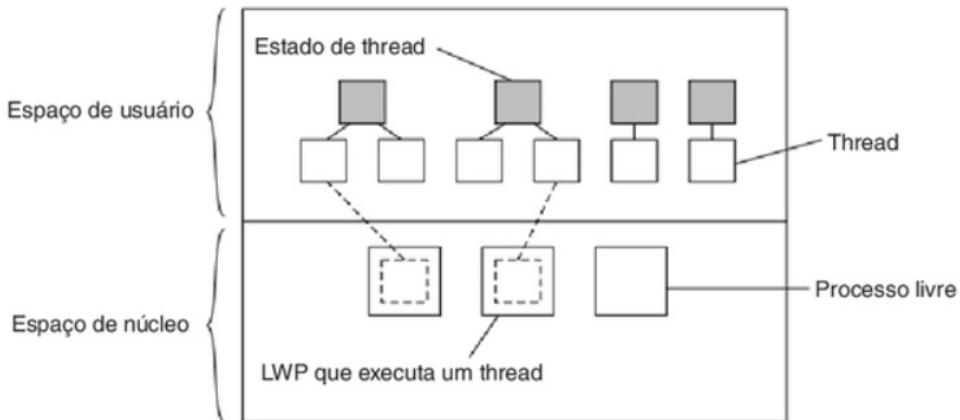
- **Threads implementadas em nível de usuário:** significa que a criação e a destruição de uma *thread* é bem menos onerosa para os recursos do sistema (principalmente o processador, que só precisa ceder acesso a alguns de seus registradores para armazenamento temporário de dados e operações que serão realizadas pela CPU), do que para a criação e destruição de processos, já que as

threads, diferentemente dos processos, são controladas em nível das aplicações dos usuários. Vamos tomar como exemplo a Figura 10: nela, os processos (Programa Usuário 1, Programa Usuário 2 e Programa Usuário 3) poderiam criar suas próprias *threads*, ou seja, dividir suas aplicações em partes menores. É importante, no entanto, perceber que se, por exemplo, o “Programa usuário 1” utilizar várias *threads* e eventualmente o mesmo seja bloqueado pelo SO, todas as suas *threads* também serão bloqueadas.

- **Threads implementadas em nível do sistema:** uma vantagem de sua utilização é que sua implementação elimina a desvantagem apontada na implementação em nível de usuário, já que a criação, destruição, sincronização, pausa, entre outras, de uma *thread* é realizada no contexto de chamadas do SO (*kernel*). Por outro lado, essa mudança de controle da *thread* pode ter um custo alto, uma vez que os recursos manipulados passam agora para a própria *thread*, acarretando prejuízo no tocante ao desempenho do sistema, que como já foi discutido, é um problema quando se lida com processos.

Tanenbaum e Steen (2007) falam também sobre os LWP, explicando que se tratam de uma implementação híbrida entre a implementação em nível de usuário e a implementação em nível do sistema. Dessa forma, os LWP executam individualmente (no contexto de um processo pesado), sendo que pode haver vários LWP por processo. A Figura 13 apresenta uma combinação dos LWP. Na Figura 13, vemos que cada LWP pode rodar sua própria *thread* (de nível de usuário) e que as *threads* de nível de usuário podem ser compartilhadas entre vários LWP.

FIGURA 13 – COMBINAÇÃO DE LWP: EM NÍVEL DE USUÁRIO E NÍVEL DO SISTEMA



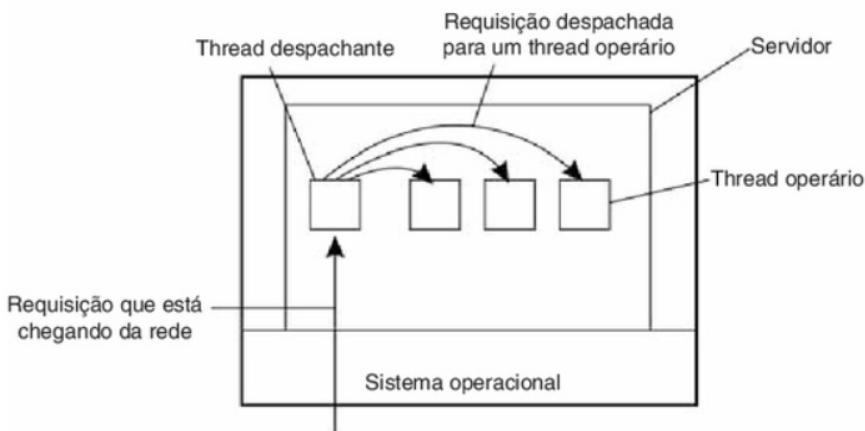
FONTE: Tanenbaum e Steen (2017, p. 44)

No tocante a sistemas distribuídos, a capacidade de manter processos operando normalmente enquanto uma de suas *threads* precisar ser bloqueada, ou entrar em pausa, é extremamente útil, pois permite manter comunicações (especialmente remotas) na forma de várias conexões lógicas ao mesmo tempo (TANENBAUM; STEEN, 2007).

Porém, mais relevante que as *threads* utilizadas no lado dos dispositivos clientes, mostradas até aqui, é o tratamento *multithread* que precisa ser dado no lado servidor para o suporte de *software* necessário aos acessos dos clientes às aplicações distribuídas. Para esse caso, Tanenbaum e Steen (2007) apresentam um modelo no qual existem dois papéis bem definidos para as *threads*, o de despachante e o de operário.

Nesse modelo da Figura 14, o **despachante** é o elemento responsável por ler as requisições que chegam ao servidor, através de uma porta específica, que após escolher um **operário** ocioso lhe repassa a requisição que chegou. Perceba que sempre existe a camada do SO desenvolvendo um papel fundamental no contexto dos sistemas distribuídos.

FIGURA 14 – SERVIDOR MULTITHREADING: MODELO DESPACHANTE E OPERÁRIO

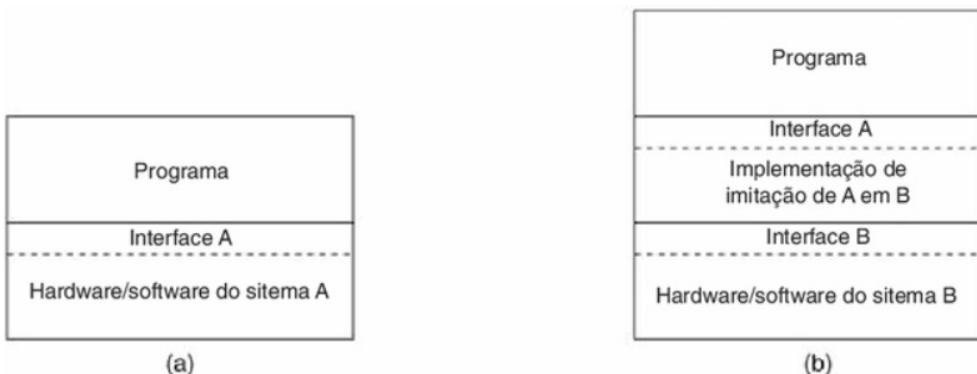


FONTE: Tanenbaum e Steen (2017, p. 47)

3 VIRTUALIZAÇÃO

Uma outra abordagem já utilizada há algum tempo, consequência dos sistemas distribuídos, é a virtualização. Em suma, ela significa que um processador pode simular um ambiente no qual pareça existir mais de um, podendo assim estender o uso dos recursos desse ambiente. Em outras palavras, virtualizar significa “imitar” (estender ou substituir) o comportamento de um outro sistema (TANENBAUM; STEEN, 2007). Veja então a Figura 15, na qual a interface de uma aplicação A — situação (a) — é virtualizada (estendida, substituída, imitada) no ambiente da aplicação B — situação (b). Pense que você pode estar em seu computador acessando um pacote do Microsoft Office que não está instalado em sua máquina, mas “virtualizado” em um servidor na rede na qual você está conectado.

FIGURA 15 – EXEMPLO DE VIRTUALIZAÇÃO



FONTE: Tanenbaum e Steen (2017, p. 48)

Virtualização é uma tecnologia que permite criar serviços de TI valiosos usando recursos que tradicionalmente estão vinculados a um determinado hardware. Com a virtualização, é possível usar a capacidade total de uma máquina física, distribuindo os recursos entre muitos usuários ou ambientes (REDHAT, 2019).



A computação em nuvem não é o mesmo que virtualização. Na verdade, a computação em nuvem é algo que você pode fazer usando a virtualização. A computação em nuvem descreve o fornecimento de recursos de computação compartilhados (software e/ou dados) sob demanda pela internet. Estando na nuvem ou não, você poderá começar virtualizando seus servidores e, em seguida, passar para a computação em nuvem para obter ainda mais agilidade e melhor autoatendimento (O QUE, 2018).

No site Vmware (VIRTUALIZAÇÃO, 2019) há uma definição de virtualização que corrobora com as definições anteriores, destacando um outro termo para a “imitação”: a “representação”, além de ressaltar os aspectos lógicos (software) e físicos dos recursos de um ambiente distribuído: virtualização é o processo de criar uma representação de algo baseada em software (ou virtual), em vez de um processo físico. A virtualização pode se aplicar a servidores, armazenamento, aplicativos e redes e é a maneira mais eficaz de reduzir as despesas de TI e, ao mesmo tempo, aumentar a eficiência e a agilidade para empresas de todos os portes.

Ainda no contexto do físico e do lógico, o site Redhat (O QUE, 2018) destaca os **hipervisores** que são softwares que definem uma fronteira entre os recursos físicos dos ambientes virtuais que os utilizam. Tais hipervisores executam em um SO (como o seu PC ou notebook, por exemplo) ou diretamente no hardware (um servidor, por exemplo).

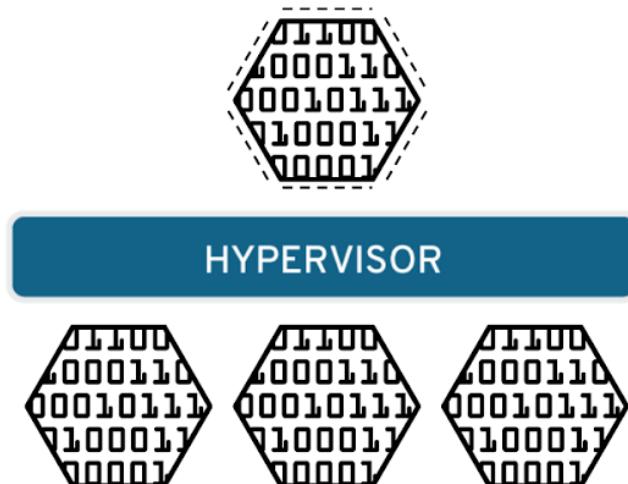
Sobre os hipervisores, Seo (2009, p. 1, grifo nosso) afirma o seguinte:

[...] em um sistema virtualizado, temos um software *host* sendo executado na máquina física, chamado *Virtual Machine Monitor* (VMM), ou *Hypervisor*. Esse software é responsável pela criação de ambientes simulados de computação, que são as VMs. Sistemas operacionais inteiros podem ser executados nessas máquinas virtuais como se estivessem sendo executados em um hardware real.

Seguem, agora, alguns tipos de virtualização (O QUE, 2018):

- **Virtualização de dados:** unifica dados distribuídos de vários locais (inclusive remotos) em uma fonte centralizada única. A Figura 16 exemplifica a virtualização de dados da seguinte forma: os dados oriundos de diferentes ambientes físicos (representados na parte inferior da figura) são, através do hipervisor, centralizados em um ambiente virtual único (parte superior da figura). Na virtualização de dados o hipervisor então faz a mediação entre as diversas fontes de dados para que o usuário possa tratá-las como se fossem apenas uma.

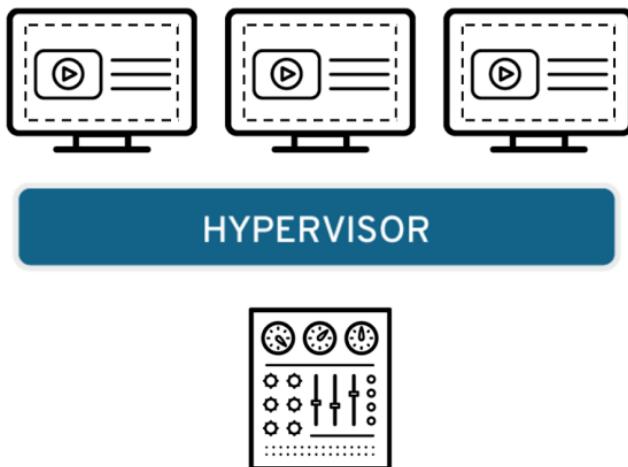
FIGURA 16 – VIRTUALIZAÇÃO DE DADOS



FONTE: <<https://www.redhat.com/cms/managed-files/data%20virtualization.png>>. Acesso em: 8 ago. 2019.

- **Virtualização de desktop:** possibilita que uma administração central (administrador ou ferramenta automatizada) instale ou utilize ambientes de desktop simulados em um considerável parque de máquinas físicas ao mesmo tempo. A Figura 17 exemplifica esta situação, demonstrando na parte inferior da figura o administrador central e na parte superior os vários ambientes de desktop simulados por intermédio do hipervisor.

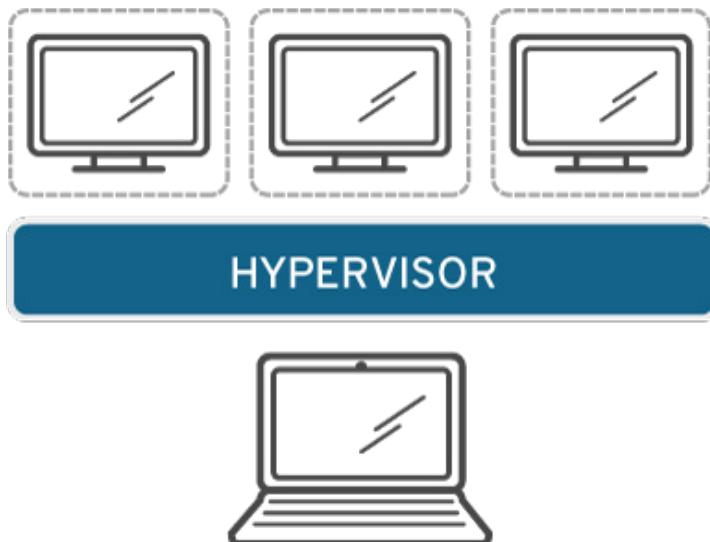
FIGURA 17 – VIRTUALIZAÇÃO DE DESKTOP



FONTE: <https://www.redhat.com/cms/managed-files/desktop%20virtualization_0.png>. Acesso em: 8 ago. 2019.

- **Virtualização de sistemas operacionais:** realizada em nível de *kernel* (já uma grande diferença da virtualização de desktop que acontece em nível de aplicação do usuário), que, como você aprendeu, representa o gerenciador fundamental das principais tarefas dos SO. Trata-se de uma boa maneira para executar plataformas diferentes (Linux e Windows, por exemplo) de maneira paralela, conforme demonstramos na Figura 18.

FIGURA 18 – VIRTUALIZAÇÃO DE SISTEMA OPERACIONAL



FONTE: <<https://www.redhat.com/cms/managed-files/operating-system-virtualization-400x321.png>>. Acesso em: 8 ago. 2019.



Não confundir a virtualização do SO (que possibilita implantar diversos SO em uma única máquina) com a virtualização de desktop (O QUE, 2018).

4 MIGRAÇÃO DE CÓDIGO

Apesar da troca de mensagens que ocorre nos sistemas distribuídos normalmente acontecer com a passagem de dados, existe a possibilidade de esses sistemas realizarem a passagem de programas (códigos), inclusive quando esses estão em execução (TANENBAUM; STEEN, 2007), que para Coulouris *et al.* (2013) trata-se de um dos grandes desafios no projeto de sistemas distribuídos. Os autores conceituam a migração de código ou código móvel, como aquele que pode ser transferido entre dispositivos e ser executado no destino.

Um exemplo deste “código móvel” são as *applets Java*, que possuem o seu código executado, normalmente, em um dispositivo cliente diferente de sua origem. E, tratando-se de programas executáveis específicos (conjunto de instruções) de uma plataforma, tem-se na JVM (*Java Virtual Machine*) uma alternativa para tornar esse código executável em muitos dispositivos clientes diferentes (COULOURIS *et al.*, 2013).

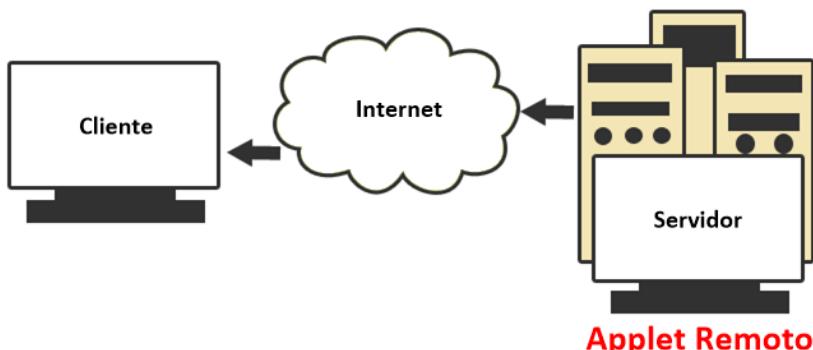


Uma *applet* é uma pequena aplicação executada em uma janela de uma aplicação (*browser/appletviewer*). Tem por finalidade estender as funcionalidades de *browsers*, adicionando som, animação, entre outros, provenientes de fontes (URLs) locais ou remotas, sendo que cada página web (arquivo .html) pode conter uma ou mais *applets*.

FONTE: <https://www.inf.pucrs.br/flash/lapro2/aula_applets.html>. Acesso em: 8 ago. 2019.

A Figura 19 apresenta o que, normalmente, descreve a existência de uma *applet*, sua localização em um servidor remoto e muitas vezes em uma plataforma diferente dos clientes que a acessam.

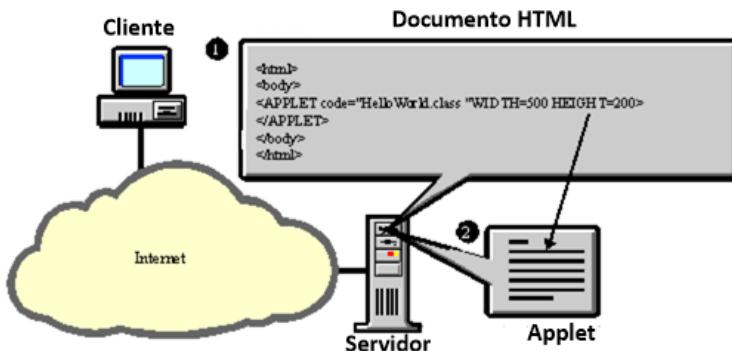
FIGURA 19 – LOCALIZAÇÃO DE UM APPLET



FONTE: <<https://www.fastlearning.in/controller/imagehub/local-applet-and-remote-applet-fastlearning.png>>. Acesso em: 2 ago. 2019.

A Figura 20 apresenta um pequeno diagrama funcional, descrito em duas etapas, que explica como uma *applet* (código de migração) é acessada e depois carregada para um dispositivo cliente.

FIGURA 20 – DIAGRAMA FUNCIONAL DE UMA APPLET



1. Cliente acessa uma página HTML com um applet incorporado a ela.
2. O bytecode Java é baixado pelo navegador e executado pelo interpretador de bytecodes integrado.

FONTE: <<https://support.novell.com/techcenter/articles/img/dnd1996080101.gif>>. Acesso em: 2 ago. 2019.

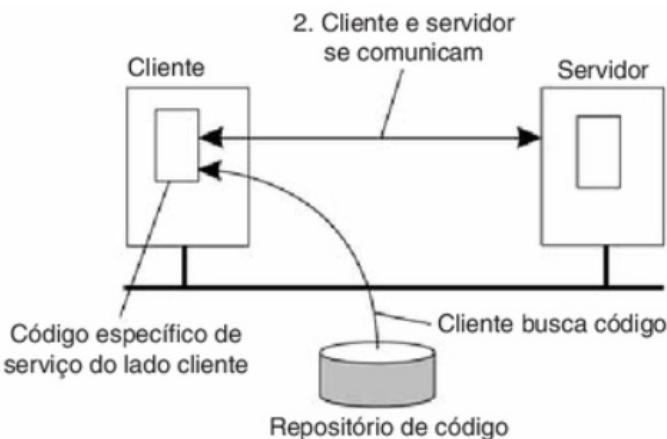
Outro exemplo, tratando-se do mais usado de migração de código, é a inclusão de trechos de código *JavaScript* em páginas web carregadas pelos navegadores clientes (COULOURIS *et al.*, 2013).



"JavaScript é a linguagem de script mais utilizada. É usada principalmente para incorporar comportamento dinâmico a páginas web (animações e melhor interatividade com o usuário). É fornecida em todos os principais navegadores" (DEITEL; DEITEL, 2017, p. 13).

Um pequeno modelo de migração de código é mostrado na Figura 21. Nela, a implementação para o cliente só é fornecida quando este se vincular ao servidor. No modelo, o cliente baixa a implementação dinamicamente, inicializa as etapas necessárias e, então, invoca o servidor (TANENBAUM; STEEN 2007).

FIGURA 21 – PRINCÍPIO DA CONFIGURAÇÃO DINÂMICA DE CLIENTE



FONTE: Tanenbaum e Steen (2007, p. 63)

RESUMO DO TÓPICO 3

Neste tópico, você aprendeu que:

- As *threads* existem no interior de um processo e compartilham entre elas os recursos deste processo, uma vez que representam uma técnica de programação importante para dividir grandes aplicações em partes menores, que podem ser executadas ao mesmo tempo.
- Existem as *threads* implementadas em nível de usuário, cuja criação e destruição é feita em nível de aplicações do usuário; e as *threads* implementadas em nível do sistema, cuja criação e destruição é realizada no contexto das chamadas do SO.
- *Multithread* é o tratamento dado do lado do servidor para o suporte de software necessário aos acessos dos clientes às aplicações distribuídas.
- Virtualização é uma abordagem que permite a um processador simular um ambiente no qual parece existir vários processadores, o que permite estender o uso dos recursos desse ambiente.
- A computação em nuvem descreve o fornecimento de recursos de computação compartilhados (software e/ou dados) sob demanda pela internet, um conceito diferente da virtualização.
- Hipervisores são softwares que definem uma fronteira entre os recursos físicos dos ambientes virtuais que os utilizam. O hipervisor é responsável pela criação de ambientes simulados de computação, como as VM (máquinas virtuais).
- São três os tipos de virtualização: virtualização de dados, virtualização de desktop e virtualização de SO.
- A migração de código, um dos grandes desafios do projeto de sistemas distribuídos, representa a possibilidade de realizarem a troca de programas entre dispositivos, mesmo que estejam em execução, de modo que possam ser executados no dispositivo destino.



- 1 Os processos são criados e destruídos. O momento e a forma pela qual eles são criados e destruídos depende do sistema operacional em consideração. Alguns sistemas trabalham com um número fixo de processos. Por exemplo, um processo para cada terminal do computador. Nesse caso, todos os processos são criados na inicialização do sistema. Eles somente são destruídos quando o próprio sistema é desligado.

FONTE: OLIVEIRA, R. S.; CARISSIMI, A. S.; TOSCANI, S. S. *Sistemas operacionais*. 4. ed. Porto Alegre: Bookman, 2010, p. 39.

Os autores lembram ainda que cada um dos processos em execução deve executar sem interferência entre eles. Dessa forma, assinale a alternativa CORRETA sobre processos e *threads*.

- a) () Um processo ou uma *thread* deve executar, sem afetar de modo intencional, malicioso ou mesmo, accidentalmente, o comportamento do outro.
- b) () Um processo ou uma *thread* pode executar, sem afetar de modo intencional, malicioso ou mesmo accidentalmente, o comportamento do outro.
- c) () Um processo é impedido pelo SO de compartilhar dados com outros processos e com suas *threads*.
- d) () Uma *thread* criada por um processo A é impedida pelo SO de compartilhar dados com outras threads criadas por outros processos.

- 2 Para a capacidade de utilizar um recurso (UCP, memória principal, armazenamento secundário por exemplo) como se houvesse mais do que um, dá-se o nome de virtualização (TANENBAUM; STEEN, 2007). Com isso, o site Vmware (O QUE, 2018) aponta que a virtualização pode permitir a implantação de cargas de trabalho mais rápidas, aumento do desempenho e a disponibilidade de recursos maiores com a automatização das atividades organizacionais, resultando em uma TI simples, barata de se operar. Com isso em mente, analise as sentenças e assinale a resposta CORRETA sobre os benefícios da virtualização:

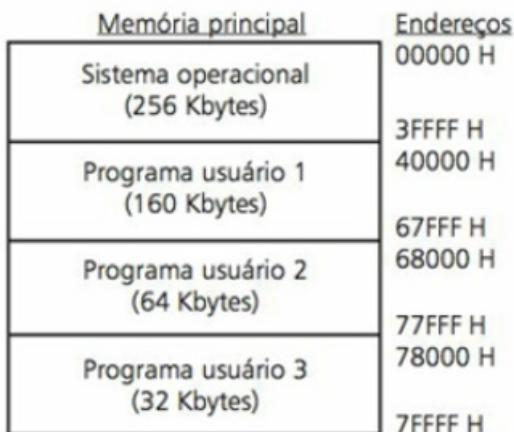
- I- Combate às despesas operacionais, porém não às de capital.
- II- Suaviza, porém, sem eliminar, o tempo de inatividade dos recursos do sistema.
- III- Agiliza a capacidade de resposta da TI.
- IV- Permite a provisão de aplicativos e recursos com mais rapidez.

Agora, assinale a alternativa que apresenta a sequência CORRETA:

- a) () As sentenças II e III estão corretas.
- b) () As sentenças III e IV estão corretas.
- c) () As sentenças II e IV estão corretas.
- d) () As sentenças I e II estão corretas.

- 3 Um processo pode ser definido como a execução individual de um programa, logo, trata-se de um componente ativo que altera, obrigatoriamente, o seu estado (durante a execução de um programa) através de chamadas de sistema ao executar esses programas (OLIVEIRA; CARISSIMI; TOSCANI, 2010). Tendo em mente a definição de processo e a sua relação com os programas, avalie a sentença de afirmação com a sua sentença de explicação relacionadas ao conceito de processo.

FIGURA – RELAÇÃO DE UM PROGRAMA E SEUS PROCESSOS



FONTE: Oliveira, Carissimi e Toscani (2010, p. 38)

I- A figura representa a execução da solução de conteúdos sob demanda disponibilizado pela Netflix (empresa estadunidense provedora de serviços de mídia) aos seus clientes para acesso via web browsers. Cada usuário acessa a solução utilizando seu login e senha, sendo que o Usuário 1 e o Usuário 3 utilizam os mesmos login e senha. A execução no servidor da Netflix, no qual cada instância do mesmo programa é executada em espaços de usuários diferentes, ocorre dessa forma.

PORQUE

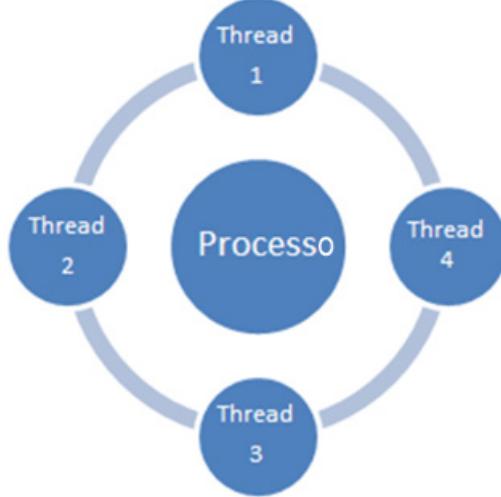
II- Cada instância de execução do programa é acessível entre si, permitindo dessa forma o compartilhamento de recursos, como memória e processador, especialmente para as instâncias dos Usuários 1 e 3, que acessando a solução com o mesmo login tem acesso a mesma área de memória e aos recursos do processador.

Agora, assinale a alternativa que apresenta a resposta CORRETA.

- a) () As sentenças I e II representam proposições falsas.
- b) () As sentenças I e II representam proposições verdadeiras, porém a II não é uma justificativa correta da I.
- c) () As sentenças I e II representam proposições verdadeiras.
- d) () A sentença I representa uma proposição verdadeira, diferentemente da proposição II que além de não ser uma justificativa correta da I é uma proposição falsa.

- 4 Em um ambiente de execução de processos computacionais é impossível desassociá-los de suas *threads* não só sob o aspecto dos SO, mas também da programação para computadores. Elas representam fluxos de execução e sempre estão associadas a um único processo, porém cada processo pode ser composto por várias *threads*. Dessa forma, avalie as afirmações sobre processos e *threads*.

FIGURA – PROCESSO E THREAD



FONTE: <https://mk0resourcesinfm536w.kinstacdn.com/wp-content/uploads/061813_1239_Multithread1.png>. Acesso em: 2 ago. 2019.

- () Um processo reserva (utiliza) um conjunto de recursos e é composto por pelo menos uma única *thread*.
- () *Threads*, de um mesmo processo, compartilham com os seus processos: espaço de endereçamento, descritores de arquivos abertos, permissões de acesso, quotas, entre outros.
- () A figura pode ser considerada um exemplo de *multithreading*.
- () Um processo reserva (utiliza) um conjunto de recursos e pode existir sem pelo menos uma única *thread*.

Agora assinale a alternativa CORRETA:

- a) () V – V – V – F.
- b) () V – V – F – F.
- c) () F – F – F – V.
- d) () V – V – F – F.

- 5 Em um ambiente de execução de processos computacionais é impossível desassociá-los de suas *threads* não só sob o aspecto dos SO, mas também da programação para computadores. Elas representam fluxos de execução e sempre estão associadas a um único processo, porém cada processo pode ser composto por várias *threads*. Dessa forma, avalie as afirmações sobre as processos e *threads*.

- () Processos permitem construir programas (porções) que parecem ser executados simultaneamente em dispositivos com um único processador.
- () *Threads* permitem construir programas (porções) que podem ser executados simultaneamente em dispositivos com mais de um processador.
- () *Threads* permitem construir programas (porções) que parecem ser executados simultaneamente em dispositivos com um único processador.
- () Processos permitem construir programas (porções) que podem ser executados simultaneamente em dispositivos com mais de um processador.

Agora, assinale a alternativa CORRETA:

- a) () V – V – V – V.
- b) () V – V – F – F.
- c) () F – V – V – F.
- d) () V – F – F – V.

TIPOS DE SISTEMAS DISTRIBUÍDOS

1 INTRODUÇÃO

Assim como existem vários tipos de SO, conforme estudamos na Unidade 1, o mesmo ocorre com os sistemas distribuídos. Existem vários tipos de sistemas distribuídos, contudo não há um consenso em sua classificação na literatura.

O formato de classificação mais comumente encontrado organiza os tipos de sistemas distribuídos em sistemas de computação distribuídos, sistemas de informação distribuídos e sistemas pervasivos distribuídos (TANENBAUM; STEEN, 2007). Neste formato, os tipos de sistemas distribuídos levam em conta a capacidade dos sistemas de realizarem computação de alto desempenho (sistemas de computação), a forma como são utilizados em aplicações em rede (sistemas de informação) e seu uso a partir de dispositivos móveis (sistemas pervasivos).

Há ainda outros paradigmas que também podem ser considerados para a tipificação de sistemas distribuídos que levam em conta sua arquitetura (sistemas distribuídos baseados em objetos), capacidade de armazenamento (sistemas de arquivos distribuídos, sistemas *peer-to-peer*, sistemas de dados multimídia) entre outros (COULOURIS *et al.*, 2013; TANENBAUM; STEEN, 2007).

Neste tópico, nos dedicaremos a estudar os tipos de sistemas de computação distribuídos (*cluster* e *grid*), e os sistemas de informação distribuídos. Os sistemas pervasivos distribuídos serão abordados no Tópico 5.

2 SISTEMAS DE COMPUTAÇÃO DISTRIBUÍDOS

Conforme já estudamos no Tópico 1, de uma maneira geral, sistemas distribuídos são definidos como um conjunto de diversos sistemas computacionais, também definidos como *hosts*, que executam diversas requisições do usuário simultaneamente, de maneira independente, passando a ideia de um ambiente único (DEITEL; DEITEL; CHOHNES, 2005). Para que isso ocorra, os sistemas distribuídos precisam implementar certas características (como transparência, escalabilidade, entre outras).

Contudo, nem sempre é possível que todas as características estejam presentes em um sistema distribuído, seja por uma questão de (reduzir) custo e/ou (aumentar) desempenho. Daí a necessidade de identificar o tipo de sistema distribuído mais adequado ao tipo de processo que se deseja realizar.

No caso dos sistemas de computação distribuídos, como já comentamos, estes são tipos de sistemas distribuídos dedicados à execução de processamento computacional de alto desempenho. Neste caso, são dois os tipos: os sistemas em *cluster* e os sistemas em grid (grade) (TANENBAUM; STEEN, 2007).

Em termos gerais, estes dois tipos de sistemas distribuídos podem ser diferenciados da seguinte forma:

- *Cluster*: “[...] consistem em um conjunto de estações de trabalho ou PCs semelhantes, conectados por meio de uma rede local de alta velocidade. Além disso, cada nó executa o mesmo sistema operacional” (TANENBAUM; STEEN, 2007, p. 10).
- Grid: “[...] costumam ser montados como federação de computadores, na qual cada sistema pode cair sob um domínio administrativo diferente, [...] no que tange a hardware, software e tecnologia de rede empregada” (TANENBAUM; STEEN, 2007, p. 10).

Na sequência, você aprenderá um pouco mais sobre estes dois tipos de sistemas computacionais distribuídos.

2.1 CLUSTER

Já antecipamos uma breve definição sobre o que é um sistema distribuído do tipo *cluster*. Para iniciarmos seu estudo mais detalhado, vamos apenas estabelecer uma definição mais completa, para que você compreenda exatamente quando um sistema distribuído se trata de um *cluster*.

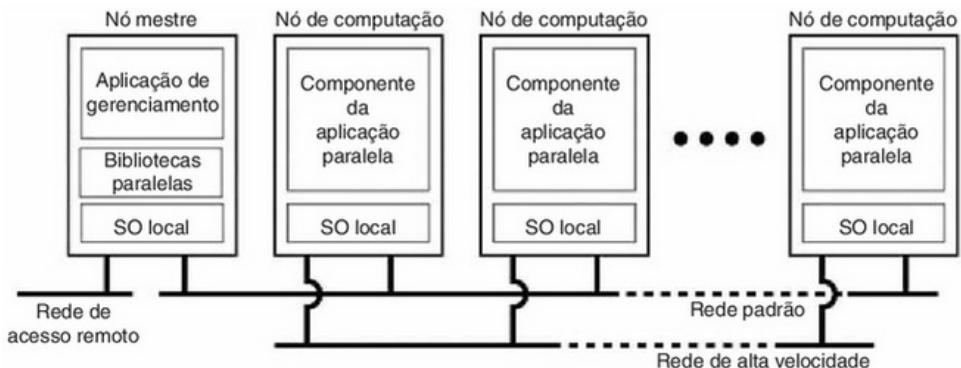
Como se trata de um sistema distribuído, um *cluster* é formado por um conjunto de sistemas computacionais conectados através de uma estrutura de rede dedicada, de alto desempenho. Sua característica primordial está na presença de um nó mestre que controla e acessa os demais nós da rede (MACHADO; MAIA, 2011; 2017; TANENBAUM; STEEN, 2007).



Um nó em uma rede nada mais é do que um computador ou, como também é conhecido, *host*.

O nó mestre é responsável por tarefas como: alocação dos nós na rede para a execução de programas paralelos, manutenção da fila de processos, manutenção da interface para os usuários. Os nós que compõem o *cluster*, por sua vez, possuem recursos próprios, incluindo SO (MACHADO; MAIA, 2011; 2017; TANENBAUM; STEEN, 2007). Veja um exemplo de sistema em *cluster* na Figura 22.

FIGURA 22 – EXEMPLO DE UM SISTEMA DISTRIBUÍDO DO TIPO CLUSTER



FONTE: Tanenbaum e Steen (2007, p. 10)

O desenvolvimento de sistemas do tipo *cluster* deu origem ao termo *clustering* ou “clusterização”. Veja como o termo pode ser definido:

Clustering - nodos de interconexão (computadores monoprocessadores ou computadores multiprocessadores) dentro de uma LAN de alta velocidade que funcionam como um único processador paralelo [...] é, em termos de arquitetura, o intermediário entre a computação distribuída e o multiprocessamento. O conjunto de nodos que forma a máquina paralela isolada é denominada **cluster**. Clustering permite que vários computadores trabalhem juntos para resolver problemas complexos, de grande porte (DEITEL; DEITEL; CHOHNES, 2005, p. 538).

Outra importante característica dos sistemas do tipo *cluster* está na possibilidade de se compartilhar dispositivos de E/S, o que, como você já estudou, garante que os recursos computacionais disponíveis sejam mais bem aproveitados (MACHADO; MAIA, 2017).

Além disso, normalmente *clusters* são implementados para uso em programação paralela. A programação paralela (ou processamento paralelo como também pode ser conhecida) é caracterizada pela situação em que um único programa é executado paralelamente por várias máquinas. A programação paralela tem como característica a redução do tempo de processamento do programa (MACHADO; MAIA, 2011; 2017; TANENBAUM; STEEN, 2007).

Uma vez que você compreendeu o que é um *cluster*, cabe agora entender como se dá seu funcionamento a partir de seus três diferentes tipos: (1) *cluster* de alto desempenho, (2) *cluster* de alta disponibilidade, e (3) *cluster* de平衡amento de carga (DEITEL; DEITEL; CHOHNES, 2005).

Em um *cluster* de alto desempenho todos os nós são utilizados na execução, por esta razão são utilizados em situações que exigem computação em grande escala, que pode ser dividida em problemas menores que serão resolvidos de forma paralela.

Quando a alta disponibilidade e a baixa ocorrência de falhas são fatores críticos, deve-se considerar a implementação de um ***cluster de alta disponibilidade***. Este tipo de *cluster* é caracterizado pela realização do trabalho por somente alguns nós, enquanto outros permanecem ociosos, como forma de backup. Assim, em caso de falha em algum nó, o trabalho é assumido imediatamente pelos nós que se encontram na reserva.

Finalmente, em um ***cluster de balanceamento de carga*** existe um nó específico responsável por realizar a tarefa de平衡ear a carga a ser distribuída ao conjunto de nós de modo que a utilização do hardware seja a mais eficiente possível. Este tipo de *cluster* é utilizado principalmente em casos de grandes volumes de requisições feitas pelos usuários.

Sistemas do tipo *cluster* tornaram-se populares por algumas razões, incluindo (DEITEL; DEITEL; CHOIFFNES, 2005; MACHADO; MAIA, 2011; 2017; TANENBAUM; STEEN, 2007):

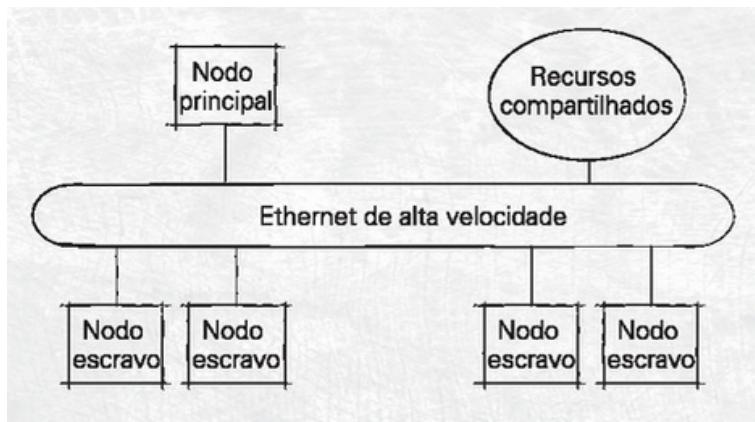
- Redução dos custos de computadores e aumento de seu desempenho, permitindo a construção de um sistema comparável a um supercomputador.
- Redução do tempo ocioso (chamado *downtime*) promovido pelo aumento da tolerância a falhas e, consequentemente, da disponibilidade.
- Escalabilidade e balanceamento de carga, uma vez que mais nós podem ser acrescentados ou eliminados para ajuste de capacidade sem que os nós restantes sejam afetados.
- No tocante da sua implementação, há vários exemplos que podem servir como forma de ilustração. A título de conhecimento, mostraremos a você exemplos de implementações do *cluster* no Linux e no Windows (DEITEL; DEITEL; CHOIFFNES, 2005).
- No caso do SO Linux, a implementação de *cluster* mais conhecida é a chamada *Beowulf*, um *cluster* de alto desempenho.



"Em 1994, o projeto *Earth and Space Sciences* (ESS), da Nasa, construiu o primeiro cluster *Beowulf* para fornecer computação paralela para resolver problemas envolvidos em aplicação da ESS" (DEITEL; DEITEL; CHOIFFNES, 2005, p. 539, grifo nosso).

De modo geral, este tipo de *cluster* pode possuir até 700 nós com Linux instalado, interconectados por uma rede Ethernet de alta velocidade. Dos nós que compõem o *cluster*, um é o nó mestre e os demais são nós seguidores. Cabe ao nó mestre controlar o acesso ao *cluster*, distribuir a carga de trabalho entre os nós seguidores, ser o responsável pelo compartilhamento de recursos, entre outras tarefas. Em um *cluster Beowulf* todos os nós ocupam uma mesma sala de modo que formem um supercomputador (DEITEL; DEITEL; CHOIFFNES, 2005). Veja na Figura 23 a organização de um *cluster Beowulf* típico.

FIGURA 23 – CLUSTER BEOWULF

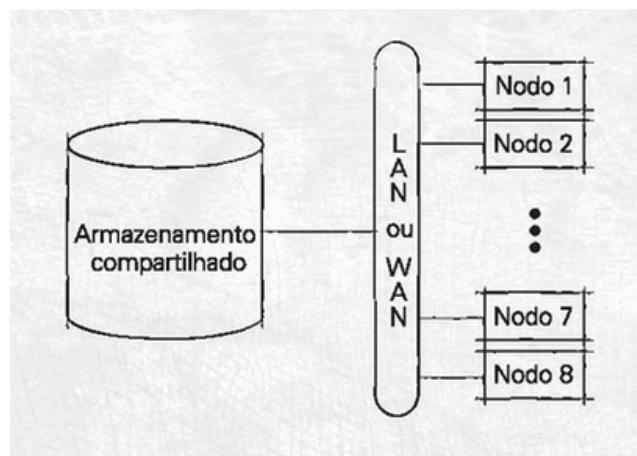


FONTE: Deitel, Deitel e Choffnes (2005, p. 540)

O outro exemplo de *cluster* que mostraremos é implementado usando o SO Windows. A partir do Windows é possível construir *clusteres* de alta disponibilidade e de平衡amento de carga, e são diferenciados pelas seguintes características (DEITEL; DEITEL; CHOFFNES, 2005):

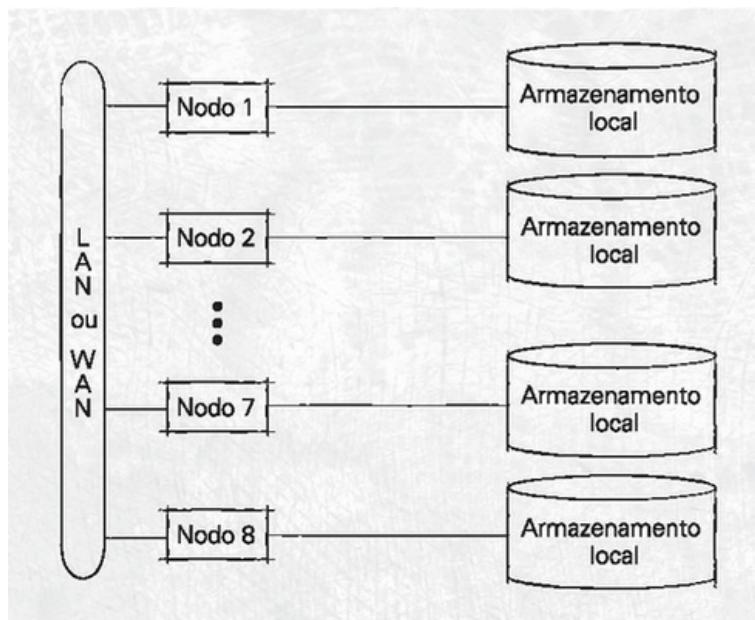
- **Cluster Windows de alta disponibilidade:** pode ser construído com até oito nós, os quais podem compartilhar um mesmo dispositivo de armazenamento ou podem cada um possuir armazenamento local. Este tipo de *cluster* pode estar conectado em uma estrutura de rede LAN (rede local) ou WAN (rede de longa distância). Veja na Figura 24 um exemplo de *cluster* Windows de alta disponibilidade com armazenamento compartilhado, e na Figura 25 um exemplo de *cluster* Windows de alta disponibilidade com armazenamento local.

FIGURA 24 – EXEMPLO DE CLUSTER WINDOWS DE ALTA DISPONIBILIDADE COM ARMAZENAMENTO COMPARTILHADO



FONTE: Deitel, Deitel e Choffnes (2005, p. 540)

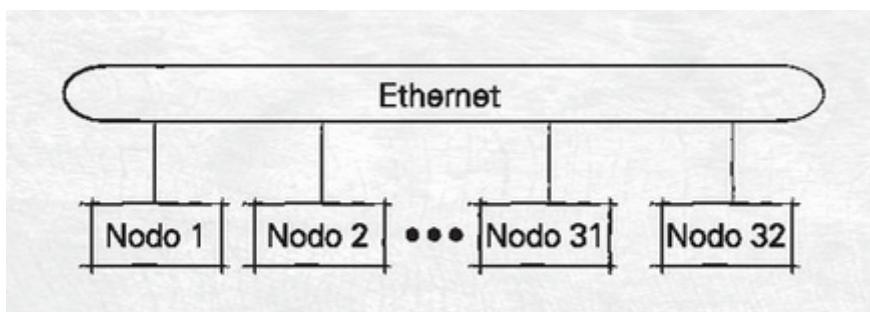
FIGURA 25 – EXEMPLO DE CLUSTER WINDOWS DE ALTA DISPONIBILIDADE COM ARMAZENAMENTO LOCAL



FONTE: Deitel, Deitel e Choffnes (2005, p. 540)

- **Cluster Windows de平衡amento de carga:** este tipo de *cluster* suporta até 32 nós que são interconectados a partir de uma estrutura Ethernet de alta velocidade. Nesta estrutura não se faz necessário o armazenamento compartilhado, uma vez que cada nó é capaz de executar seus serviços de forma independente dos demais. A Figura 26 mostra a estrutura do *cluster* Windows de balanceamento de carga.

FIGURA 26 – EXEMPLO DE CLUSTER WINDOWS DE BALANCEAMENTO DE CARGA



FONTE: Deitel, Deitel e Choffnes (2005, p. 541)

2.2 GRID (GRADE)

Ao iniciarmos esta etapa de seus estudos, fornecemos uma definição dos sistemas distribuídos do tipo grid que os qualificava como uma “federação de computadores”, cada um sob um domínio administrativo diferente. Eventualmente, esta pode se tratar de uma definição um pouco abrangente sobre um sistema em grid, mas ainda assim válida. É importante então compreendermos o que o autor pretendia associar com a expressão “federação de computadores” e como relacionar isso aos sistemas de computação distribuída.

Segundo o dicionário Aurélio (2005), uma federação consiste de “uma política entre estados, sob um governo central” ou ainda “Associação, aliança”. Fazendo a relação com o contexto tecnológico, uma “federação de computadores” reúne então diferentes sistemas computacionais, no caso distribuídos em uma rede de computadores, com um objetivo em comum. Faz sentido, não faz?

Então, estamos falando de um sistema distribuído assim como o *cluster* que você acabou de estudar. Mas então, de fato, qual a diferença entre estes dois tipos de sistemas? A diferença está na organização dos sistemas computacionais que compõem o sistema distribuído: enquanto em um *cluster* existe a figura do nó mestre centralizado, em um sistema em grid é privilegiada a “federação de computadores”, ou seja, a colaboração pública (DEITEL; DEITEL; CHOFFNES, 2005).

Você deve estar se perguntando: “como isso ocorre?”, “como ocorre a colaboração pública?”. Lembre-se que, com o barateamento dos computadores e o advento das redes a partir da década de 1980, temos disponíveis, atualmente, um número considerável de computadores em utilização ao redor do mundo. Computadores em casa, no trabalho, nas escolas e universidades. A característica em comum destes computadores é o que chamamos de “capacidade computacional desperdiçada”. Ou seja, “[...] computadores pessoais ficam ociosos nas residências enquanto as pessoas estão no trabalho, e computadores de trabalho ficam ociosos enquanto as pessoas estão em casa” (DEITEL; DEITEL; CHOFFNES, 2005, p. 545).

Uma questão fundamental em um sistema de computação em grade é que recursos de diferentes organizações são reunidos para permitir a colaboração de um grupo de pessoas ou instituições. Tal colaboração é realizada sob a forma de uma **organização virtual** (TANENBAUM; STEEN, 2007, p. 11).

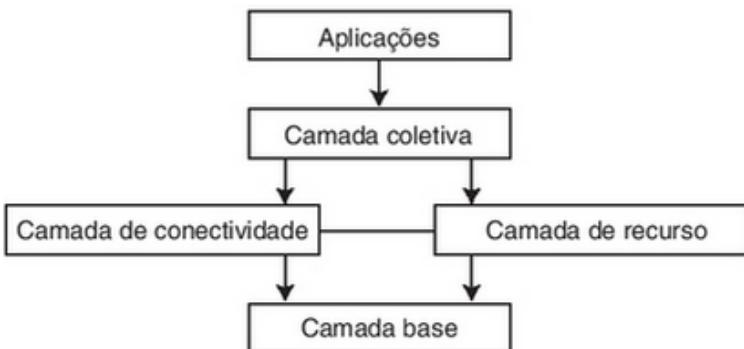
O que define um sistema de computação em grid é a interligação entre diferentes recursos computacionais (por exemplo, o seu notebook em sua casa, o computador de uma pessoa qualquer em seu trabalho, um servidor em uma universidade) distribuídos em uma rede, com a finalidade de resolver problemas complexos de computação (DEITEL; DEITEL; CHOFFNES, 2005).

Dentre as características dos sistemas distribuídos em grid merecem destaque (DEITEL; DEITEL; CHOFFNES, 2005; TANENBAUM; STEEN, 2007):

- Por se tratar de um sistema distribuído, os recursos em uma grid são acessados de forma transparente pelos usuários.
- O principal objetivo dos sistemas em grid é atingir alto desempenho, conseguido a partir da utilização das sobras da capacidade computacional dos vários sistemas computacionais que a compõe.
- Oferece maior escalabilidade que um *cluster*.
- Diferentemente dos *clusters* que são sistemas homogêneos (são formados por máquinas iguais), sistemas em grid são sistemas heterogêneos (são formados por máquinas diferentes, não havendo restrições de hardware, SO, redes, entre outros).

Outro aspecto que merece destaque nos sistemas distribuídos em grid é sua arquitetura, dividida em cinco camadas: camada-base, camada de conectividade, camada de recursos, camada coletiva e camada de aplicações, conforme demonstra a Figura 27.

FIGURA 27 – ARQUITETURA DE UM SISTEMA DISTRIBUÍDO EM GRID



FONTE: Tanenbaum e Steen (2007, p. 11)

Cada camada pode ser definida da seguinte forma (DEITEL; DEITEL; CHOHNES, 2005; TANENBAUM; STEEN, 2007):

- **Camada-base:** camada mais inferior na arquitetura de um sistema em grid é responsável por fornecer as interfaces de acesso a recursos físicos em um sistema computacional específico.
- **Camada de conectividade:** é a camada responsável por implementar protocolos que permitem a comunicação entre os diferentes recursos na rede, garantindo que esta comunicação aconteça de forma segura e confiável.
- **Camada de recursos:** responsável por gerenciar os recursos da grid individualmente. Permite o compartilhamento a partir das funções fornecidas pela camada de conectividade.
- **Camada coletiva:** esta camada coordena os recursos distribuídos, manipulando o acesso aos mesmos, fazendo o escalonamento de tarefas, a replicação de dados, entre outros.
- **Camada de aplicação:** nível mais alto, a camada de aplicação contém as aplicações que são utilizadas no ambiente distribuído.



O projeto SETI@home (*Search for Extraterrestrial Intelligence at home*) (setiathome.ssl.berkeley.edu) é uma implementação popular de computação em grade. O SETI@home habilita indivíduos a participar de um esforço científico que procura vida inteligente em outros lugares do universo. Os computadores dos participantes, quando não estão em uso, descarregam dados que representam sinais do espaço sideral do servidor SETI@home, analisam esses dados e retornam os resultados ao servidor SETI@home (DEITEL; DEITEL; CHOFFNES, 2005, p. 546, grifo nosso).

3 SISTEMAS DE INFORMAÇÃO DISTRIBUÍDOS

O contexto dos sistemas de informação distribuídos é apresentado por Tanenbaum e Steen (2007) e os relaciona ao âmbito empresarial, como resultado da construção de uma infraestrutura que promoveu a integração entre sistemas.

Neste sentido, os autores abordam dois tipos de sistemas de informação distribuídos, os sistemas capazes de executar transações distribuídas, denominados sistemas de processamento de transações, e aqueles responsáveis pela integração de aplicações empresariais (EIA – *Enterprise Application Integration*).

Os sistemas de processamento de transações são oriundos de uma arquitetura reconhecida como cliente/servidor. Nesta arquitetura, o servidor, que normalmente hospeda um banco de dados, executa as aplicações e as disponibiliza para seus clientes (computadores remotos). Em outras palavras, um computador cliente pode encaminhar uma requisição ao servidor, que executa uma operação específica em seu banco de dados para atendê-la, retornando uma resposta ao cliente. Ao conjunto “requisição-execução-resposta”, especialmente quando envolvendo um banco de dados, damos o nome de **transação**.



Os sistemas de processamento de transações estão diretamente relacionados aos bancos de dados, isso porque, “na prática, operações em um banco de dados costumam ser realizadas sob a forma de transações” (TANENBAUM; STEEN, 2007, p. 12).

A principal característica do processamento de transações está relacionada à seguinte condição: ou todas as operações que constituem uma transação (chamadas diretivas ou primitivas) são executadas ou nenhuma é. No entanto, há ainda outras propriedades que caracterizam uma transação: atômicas, consistentes, isoladas e duráveis.

- **Atômicas:** “para o mundo exterior, a transação acontece como se fosse indivisível” (TANENBAUM; STEEN, 2007, p. 13). A característica de atomicidade garante o que acabamos de mencionar: ou uma transação acontece completamente, ou não acontece. E quando acontece é indivisível.
- **Consistentes:** “a transação não viola invariantes de sistema” (TANENBAUM; STEEN, 2007, p. 13), ou seja, situações previstas no sistema, consideradas invariáveis, deverão ser mantidas mesmo após a ocorrência da transação. Para ficar mais claro, imagine uma situação em um supermercado, em que um caixa que ficou sem notas de cinco reais, envia para um outro caixa uma nota de vinte reais para trocar por quatro notas de cinco. Durante a transação o primeiro caixa terá seu valor reduzido em vinte reais, porém este valor voltará a ser o mesmo de antes da transação quando a troca se efetivar.
- **Isoladas:** “transações concorrentes não interferem umas com as outras” (TANENBAUM; STEEN, 2007, p. 13), o que significa que se mais de uma transação for executada simultaneamente, seu resultado se apresentará como se elas tivessem sido executadas de forma sequencial.
- **Duráveis:** “[...] uma vez comprometida uma transação, as alterações são permanentes” (TANENBAUM; STEEN, 2007, p. 13). Isso significa dizer que, não importa o que ocorra, nem mesmo uma falha é capaz de desfazer os resultados ou provocar a perda de uma transação.



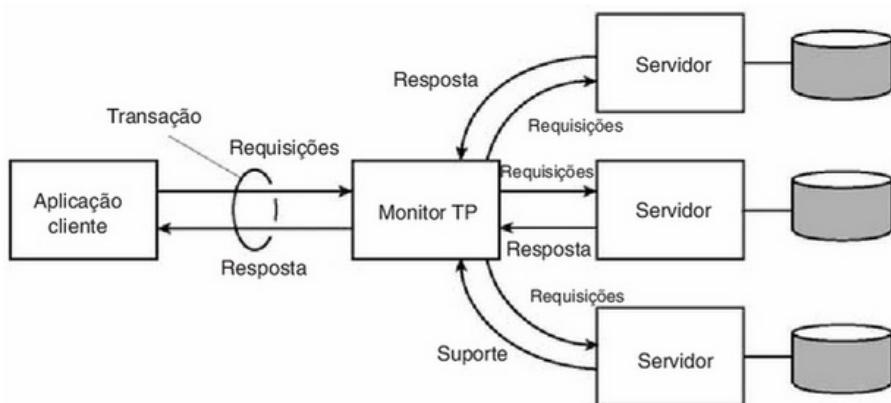
As características de uma transação que você acabou de ver são normalmente identificadas pela sigla ACID, formada pelas iniciais de cada uma das características.

Inicialmente, as transações ocorriam em um único banco de dados. Posteriormente, o contexto de transação distribuída ou transação aninhada, passou a envolver um conjunto de subtransações executadas em paralelo em computadores diferentes para se obter, entre outras coisas, melhoria de desempenho.

Quando os sistemas de middleware empresarial começaram, o componente que manipulava transações distribuídas, ou aninhadas, formava o núcleo para a integração de aplicações no nível do servidor ou do banco de dados. Esse componente era denominado **monitor de processamento de transação**, ou, de forma abreviada, **monitor TP**. Sua principal tarefa era permitir que uma aplicação acessasse vários servidores/bancos de dados [...] (TANENBAUM; STEEN, 2007, p. 14, grifo do original).

Veja na Figura 28 o funcionamento de um sistema de processamento de transações distribuído.

FIGURA 28 – SISTEMA DE PROCESSAMENTO DE TRANSAÇÕES DISTRIBUÍDO

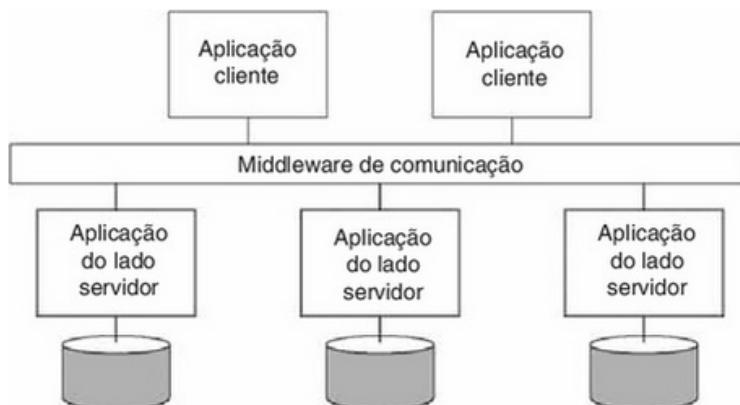


FONTE: Tanenbaum e Steen (2007, p. 14)

A **Integração de Aplicações Empresariais** (EIA) passou a ser necessária à medida que as aplicações se desvinculavam dos bancos de dados. Mais especificamente: “[...] componentes de aplicação deveriam poder se comunicar diretamente uns com os outros, e não apenas por meio do comportamento requisição/resposta [...]” (TANENBAUM; STEEN, 2007, p. 14) como acontecia com os sistemas de processamento de transações que você acabou de estudar.

Desta forma, uma série de diferentes modelos de comunicação surgiu, compondo um *middleware* que possibilitasse a troca de informações diretamente entre as aplicações clientes e o servidor. Veja como isso funciona na Figura 29.

FIGURA 29 – MIDDLEWARE DE COMUNICAÇÃO PARA A INTEGRAÇÃO DE APlicações EMPRESARIAIS



FONTE: Tanenbaum e Steen (2007, p. 14)

Dentre os tipos de *middleware* podemos citar (TANENBAUM; STEEN, 2007, p. 14):

- Chamada de procedimento remoto (RPC - *Remote Procedure Calls*): neste *middleware* um componente de aplicação pode enviar uma requisição a um outro componente a partir de uma chamada de procedimento local.
- Invocações de método remoto (RMI - *Remote Method Invocations*): basicamente o mesmo que um RPC, contudo funcionando com objetos ao invés de aplicações.
- *Middleware* orientado a mensagem (MOM - *Message-oriented Middleware*): aqui as aplicações enviam mensagens a pontos lógicos de contato, cabendo ao *middleware* garantir que as mensagens sejam entregues às aplicações.

RESUMO DO TÓPICO 4

Neste tópico, você aprendeu que:

- Os sistemas de computação distribuídos dedicam-se à execução de processamento computacional de alto desempenho, e são dois os tipos: os sistemas em cluster e os sistemas em grid (grade).
- Um *cluster* é formado por um conjunto de sistemas computacionais conectados através de uma estrutura de rede dedicada, de alto desempenho, apresentando um nó mestre que controla e acessa os demais nós da rede.
- A clusterização permite a programação paralela, quando um único programa é executado paralelamente por várias máquinas. A programação paralela tem como característica a redução do tempo de processamento do programa.
- Dentre as razões para o uso do *cluster* destacam-se: redução dos custos dos computadores e aumento do seu desempenho, redução do tempo ocioso com aumento da tolerância a falhas, escalabilidade e balanceamento de carga.
- Uma grid é caracterizada por reunir diferentes sistemas computacionais, distribuídos em uma rede de computadores, com um objetivo em comum.
- Em uma grid não há um nó mestre, mas sim a “colaboração pública”.
- Dentre as características da computação em *grid* destacam-se: transparência de acesso, alto desempenho, maior escalabilidade, são sistemas heterogêneos.
- Os sistemas de informação distribuídos são resultado da construção de uma infraestrutura que promoveu a integração entre sistemas.
- Os sistemas de processamento de transações são oriundos de uma arquitetura reconhecida como cliente/servidor.
- A Integração de Aplicações Empresariais (EIA) passou a ser necessária à medida que as aplicações se desvinculavam dos bancos de dados.



- 1 A implementação das características de um sistema distribuído passa pela relação custo/desempenho. Com isso, é importante ter em mente o que se deseja destacar nas funcionalidades desse tipo de sistema, sendo que de uma maneira geral pode-se escolher entre os sistemas em *Cluster* e os sistemas em *Grid*. Vale ressaltar que apesar disso, não há um consenso definitivo para estabelecer essa classificação. Dessa forma, avalie as sentenças e indique a alternativa CORRETA sobre essa classificação:
- a) () A configuração baseada em *Cluster* é representada por um conjunto de dispositivos semelhantes rodando sobre uma rede local de alta velocidade, e nessa configuração cada *host* pode executar um SO diferente.
 - b) () As configurações em *Grid* costumam ser montadas como a união de vários computadores como um só dispositivo, formando uma espécie de computador coletivo, e na qual cada sistema (hardware, software e infraestrutura de rede) pode ser administrado de maneira independente.
 - c) () A configuração baseada em *Grid* é representada por um conjunto de dispositivos semelhantes rodando sobre uma rede local de alta velocidade, e nessa configuração cada *host* executa o mesmo SO.
 - d) () A configuração baseada em *Cluster* costuma ser montada como a união de vários computadores como um só dispositivo, formando uma espécie de computador coletivo, e na qual cada sistema (hardware, software e infraestrutura de rede) pode ser administrado de maneira independente.
- 2 A implementação das características de um sistema distribuído passa pela relação custo/desempenho. Com isso, é importante ter em mente o que se deseja destacar nas funcionalidades desse tipo de sistema, sendo que, de uma maneira geral, pode-se escolher entre os sistemas em *Cluster* e os sistemas em *Grid*. Vale ressaltar que, apesar disso, não há um consenso definitivo para estabelecer essa classificação. Dessa forma, avalie as sentenças sobre a popularização dos sistemas baseados em *Cluster*, classificando V para as sentenças verdadeira e F para as falsas:
- () Boa relação no tocante custos/desempenho, com custos reduzidos, apesar do aumento de desempenho não ser significativo. Aqui o custo passa a ser mais relevante permitindo a construção de um sistema comparável a um supercomputador.
 - () A relação escalabilidade/balanceamento de carga é considerável, permitindo acréscimo, e/ou eliminação, de *hosts* facilitando assim o *tuning* de sua capacidade sem afetar o sistema como um todo.
 - () Boa relação no tocante custos/desempenho, com custos reduzidos e aumento de desempenho significativo. Aqui a custo/desempenho tem a mesma relevância.
 - () A relação escalabilidade/balanceamento de carga é considerável, permitindo acréscimo, e/ou eliminação, de *hosts* facilitando assim o *tuning* de sua capacidade, porém com uma certa degradação do sistema como um todo.

Agora, assinale a alternativa CORRETA:

- a) () V – F – F – V.
- b) () F – V – V – F.
- c) () V – F – V – F.
- d) () V – V – F – F.

3 A implementação das características de um sistema distribuído passa pela relação custo/desempenho. Com isso é importante ter em mente o que se deseja destacar nas funcionalidades desse tipo de sistema, sendo que de uma maneira geral pode-se escolher entre os sistemas em *Cluster* e os sistemas em *Grid*. Vale ressaltar que, apesar disso, não há um consenso definitivo para estabelecer essa classificação. Dessa forma, avalie as sentenças relacionadas as características dos sistemas em *Grid*:

- () Esse tipo de sistema objetiva, de maneira secundária, o alto desempenho, alcançado com o compartilhamento das fatias de recursos do conjunto computacional. Ele tem como foco principal uma arquitetura heterogênea para a convivência de vários sistemas diferentes.
- () Esse tipo de sistema objetiva, acima de tudo, o alto desempenho, alcançado com o compartilhamento das fatias de recursos, do conjunto computacional, dos vários dispositivos conectados que o compõe.
- () Do ponto de vista dos sistemas que o compõem são também homogêneos (formados por máquinas iguais) como os seus irmãos *clusteres*, permitindo a utilização do mesmo SO em todos os hosts.
- () Do ponto de vista dos sistemas que o compõem são heterogêneos (porém, formados por uma quantidade maior de máquinas iguais) como os seus irmãos *clusteres*, permitindo a utilização do mesmo SO em todos os hosts.

A partir das definições fornecidas, estão CORRETAS as assertivas:

- a) () II e IV.
- b) () II e III.
- c) () Somente a I.
- d) () Somente a II.

4 A construção de um sistema distribuído pode ser alcançada por intermédio de, basicamente, dois tipos, os sistemas em *Cluster* e os sistemas em *Grid*. Ambos carregam um fator custo/desempenho que deve ser considerado no desenvolvimento de suas funcionalidades. Sendo que, apesar disso, não há um consenso estabelecido e definitivo em relação a essa tipificação. Tendo isso em mente, avalie as sentenças relacionadas às características dos sistemas em *Grid*:

- () Existe uma camada responsável pela coordenação dos recursos. Ela manipula o acesso a esses recursos, promove o escalonamento de tarefas, além de lidar com dados replicados, trata-se da camada coletiva.

- () Além das cinco camadas conceituais da arquitetura de um sistema distribuído em grid, existe uma sexta, implícita, que envolve toda as outras para criar um nível de abstração que facilite o acesso dos usuários às funcionalidades do sistema.
- () Existe uma camada responsável pela coordenação dos recursos. Ela manipula o acesso a esses recursos, promove o escalonamento de tarefas, além de lidar com dados replicados, trata-se da camada de recursos.
- () A camada mais distante das interações com o usuário é a Camada-base, ela tem a importante função de gerenciar a comunicação de camadas superiores, com os pontos de conexão de acesso aos recursos físicos de um dado sistema computacional.

A partir das definições fornecidas, estão CORRETAS as assertivas:

- a) () III e IV.
b) () I e IV.
c) () Somente a IV.
d) () II e III.

5 Existe um conjunto de elementos que relaciona os sistemas de informação distribuídos ao contexto empresarial, tendo como causa o nascimento de uma infraestrutura que propiciou a integração entre sistemas. Nesse contexto, autores apresentam uma tipificação para os sistemas de informação distribuídos, sendo assim, assinale a alternativa CORRETA em relação a essa tipificação:

- a) () Existem sistemas capazes de trabalhar com transações distribuídas, que tiveram origem no modelo cliente/servidor. Nesse modelo, geralmente existe a figura de um servidor que mantém, normalmente, um banco de dados, é responsável pelas aplicações e pela sua disponibilização às máquinas clientes, tais sistemas são denominados sistemas de integração de aplicações empresariais.
- b) () Existem sistemas capazes de trabalhar com transações distribuídas, que tiveram origem no modelo cliente/servidor. Nesse modelo, geralmente existe a figura de um servidor que mantém, normalmente, um banco de dados, é responsável pelas aplicações e pela sua disponibilização às máquinas clientes, tais sistemas são denominados sistemas de processamento de transações.
- c) () A partir de um dado momento o ambiente empresarial necessitou de uma maior integralidade entre as suas aplicações, e nesse momento tais aplicações passam a atuar de uma maneira mais independente dos bancos de dados. Aqui passa-se à segunda geração dos sistemas de processamento de transações.
- d) () A partir de um dado momento o ambiente empresarial necessitou de uma maior integralidade entre as suas aplicações, e nesse momento tais aplicações passam a necessitar de componentes com a capacidade de comunicação diretamente entre si, o comportamento básico requisição/resposta evoluiu. Aqui passa-se à segunda geração dos sistemas de processamento de transações.

COMPUTAÇÃO MÓVEL E UBÍQUA

1 INTRODUÇÃO

Chegando a este último tópico desta nossa unidade, você já compreendeu o que são os sistemas distribuídos e sua finalidade. Especialmente em relação a sua finalidade, vimos que, além de permitirem a realização de várias tarefas simultaneamente, sua distribuição em estruturas de redes de computadores permitiu a evolução de um novo formato de comunicação.

Na mesma mão desta revolução, nas comunicações proporcionada pelas redes, em especial a internet, vimos também a evolução e integração de dispositivos eletrônicos de computação, capazes de cooperarem entre si como sistemas distribuídos proporcionando ao usuário acesso rápido e transparente a serviços e aplicações (ARAUJO, 2003 *apud* CIRILO, 2006; COULOURIS *et al.*, 2013).

Em relação aos dispositivos eletrônicos de computação, podemos entender que sua miniaturização, tornando-os portáteis, e seu barateamento, tornando-os acessíveis a todos os públicos, contribuíram para que se fizessem cada vez mais presentes na rotina dos indivíduos. Assim, nos dias de hoje, não é incomum encontrarmos pessoas portando aparelhos como notebooks, *smartphones*, câmeras de vídeo e digitais, relógios de pulso inteligentes, e até mesmo dispositivos incorporados em outros aparelhos como geladeiras e automóveis (COULOURIS *et al.*, 2013).

Perceba então que:

À medida que os equipamentos se tornam menores, fica mais fácil levá-los conosco ou vesti-los, e podemos incorporá-los em muitas partes do mundo físico [...]. E, à medida que a conectividade sem fio se torna predominante, podemos conectar melhor esses novos e pequenos dispositivos uns com os outros, com computadores pessoais e com servidores convencionais (COULOURIS *et al.*, 2013, p. 817).

A partir de características como estas que acabamos de apontar, a miniaturização dos equipamentos e sua conexão em redes (especialmente sem fio) surgiram dois conceitos de grande interesse: a **computação móvel** e a **computação ubíqua**, que estudaremos neste tópico. Além destes dois conceitos há ainda o conceito de **computação pervasiva** (já mencionado no Tópico 4), normalmente considerada um sinônimo da computação ubíqua, mas que, você perceberá, existem diferenças entre ambas, e por isso também abordaremos este terceiro conceito.



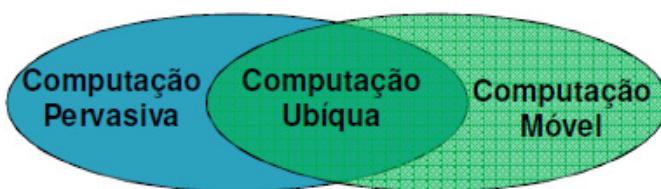
Embora as expressões computação ubíqua, computação pervasiva, computação nomádica, computação móvel e outras tantas, sejam usadas muitas vezes como sinônimos, elas são diferentes conceitualmente e empregam distintas ideias de organização e distinto gerenciamento dos serviços computacionais (ARAÚJO, 2003 *apud* SANTAELLA, 2013, p. 16).

Inicialmente, os três conceitos que seguirão nossos estudos podem ser apresentados da seguinte maneira:

- **Computação ubíqua:** “[...] entende-se por ubiquidade a coordenação de dispositivos inteligentes, móveis e estacionários para prover aos usuários acesso imediato e universal à informação e novos serviços, de forma transparente, visando aumentar as capacidades humanas” (ARAÚJO, 2003 *apud* SANTAELLA, 2013, p. 16).
- **Computação móvel:** “[...] se define pela possibilidade de movimentação física humana levando junto consigo serviços computacionais. Isso significa que a computação se torna uma atividade que pode ser transportada para qualquer lugar a qualquer hora” (ARAÚJO, 2003 *apud* SANTAELLA, 2013, p. 16).
- **Computação pervasiva:** “[...] significa que o computador está embarcado no ambiente de forma invisível para o usuário. Desse modo, o computador tem a capacidade de obter informação do ambiente no qual ele está embarcado e utilizá-la para dinamicamente construir modelos computacionais” (ARAÚJO, 2003 *apud* SANTAELLA, 2013, p. 17).

Perceba que, de modo geral, a computação móvel está relacionada ao uso e conexão dos dispositivos móveis; no outro extremo, a computação pervasiva se encarrega de controlar, configurar e ajustar as aplicações de modo a atender às necessidades tanto do dispositivo quanto do usuário; por fim, a computação ubíqua, colhe os benefícios de ambas, sendo a responsável por integrar os dispositivos computacionais com o mundo físico (ARAÚJO, 2003 *apud* SANTAELLA, 2013; COULOURIS *et al.*, 2013). Veja na Figura 30 a relação entre os três conceitos:

FIGURA 30 – RELAÇÃO ENTRE COMPUTAÇÃO PERVASIVA, UBÍQUA E MÓVEL



FONTE: Araújo (2013 *apud* CIRILO, 2006)

2 COMPUTAÇÃO MÓVEL

Você deve lembrar que a década de 1980 contribuiu para a popularização dos computadores, principalmente pelo lançamento dos computadores pessoais, mais baratos e, especialmente, menores e mais leves. Uma das características destas máquinas é que já era possível que se conectasseam entre si a partir de um modem e uma linha telefônica.

À medida que os anos seguiram, os computadores continuaram evoluindo, diminuindo em tamanho e melhorando suas funcionalidades e desempenho, surgindo então os notebooks, e mais recentemente netbooks e tablets.

Estes dispositivos praticamente consolidaram o conceito de computação móvel, surgido exatamente da ideia de que seria interessante se os usuários pudessem “[...] carregar seus computadores pessoais e manter certa conectividade com outras máquinas” (COULOURIS *et al.*, 2013, p. 818), já que suportam diferentes formas de conexão sem fio, como Wi-Fi (*Wireless Fidelity*), redes de celulares e *bluetooth*.

Nesse sentido, a evolução caminhou em direção à chamada computação de mão (*handheld computing*), caracterizada pelo uso de aparelhos de dimensões reduzidas, que cabem na mão. Os mais famosos aparelhos desta categoria são nossos *smartphones* além dos PDA (*Personal Digital Assistants*). Veja a diferença entre eles na Figura 31, que mostra um PDA *Palm Pilot* (à esquerda) e um *smartphone* iPhone (à direita).

FIGURA 31 – PDA VS SMARTPHONE



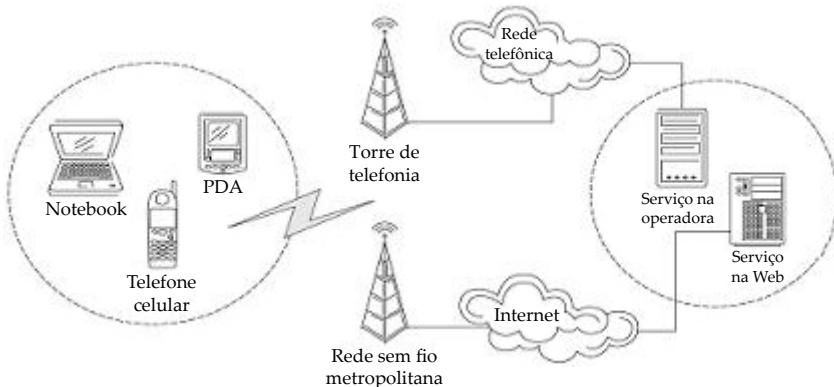
FONTE: <<https://zdnet4.cbsistatic.com/hub/i/r/2017/11/06/4914e380-a42a-4368-bc86-dfe3867dc9a/resize/770xauto/eb985348a246674e81f24b3540cad2fc/palm-vs-iphone-se.jpg>>. Acesso em: 5 ago. 2019.



Os *smartphones* e os PDAs são capazes de executar muitos tipos diferentes de aplicações; mas, comparados aos notebooks, têm menor tamanho e bateria de maior duração, possuem um poder de processamento correspondentemente limitado, uma tela menor e outras restrições de recursos (COULOURIS *et al.*, 2013, p. 818, grifo nosso).

Perceba que “a portabilidade de muitos desses dispositivos, junto a sua capacidade de se conectar convenientemente com redes em diferentes lugares, tornam a computação móvel possível” (COULOURIS *et al.*, 2013, p. 10). Veja na Figura 32 como isso é possível.

FIGURA 32 – AMBIENTE DE COMPUTAÇÃO MÓVEL



FONTE:<https://www.devmedia.com.br/imagens/Nova%20pasta%20%282%29/pb_25_05_09_pic01.JPG>. Acesso em: 5 ago. 2019.

Logo, fica mais fácil agora compreender que somente teremos um ambiente de computação móvel a partir do momento que existirem diferentes dispositivos se comunicando entre si através de uma conexão de rede sem fio. Em um cenário como este é possível ao usuário executar suas tarefas computacionais (como acessar sua conta de e-mail, por exemplo), à medida que se desloca de um lugar para outro (ADELSTEIN *et al.*, 2005 *apud* CIRILO, 2006).

Perceba que a própria natureza de um ambiente de computação móvel exige um modelo de dispositivo específico, cujas características envolvem, dentre outras, limitações em relação a sua fonte de energia e recursos computacionais. Entenda melhor estas características (COULOURIS *et al.*, 2013):

- **Energia limitada:** dispositivos portáteis móveis, como *smartphones* e notebooks, por exemplo, dependem de uma bateria para funcionar. A comunicação sem fio demanda muita energia para executar ações como o envio e recebimento de mensagens, e até mesmo quando estes dispositivos estão em modo de espera o consumo é considerável. Desta forma, quanto menor e mais leve o dispositivo, menor será a capacidade necessária de sua bateria.
- **Restrições de recurso:** o consumo de energia dos dispositivos móveis vem lhes causar outro impacto: limitação dos recursos computacionais (por exemplo: velocidade do processamento e capacidade de armazenamento). Melhoria dos recursos computacionais necessariamente implica em maior consumo de energia. Outra questão relacionada à limitação de recursos é consequência do próprio tamanho reduzido dos dispositivos, o que ocasiona em menor número de transistores nos processadores, por exemplo.
- **Sensores e controladores:** sensores e controladores são elementos que fazem parte dos dispositivos móveis. Os sensores são responsáveis por medir valores físicos e fornecê-los a algum software (por exemplo: sua posição geográfica em um aplicativo de GPS – *Global Positioning System*), já os controladores são controlados por software e afetam o mundo físico (por exemplo: um ar condicionado programável).

Convém ainda chamar atenção para uma limitação importante relacionada à computação móvel: a percepção e reconhecimento de contexto. Por definição “o contexto de uma entidade (pessoa, lugar ou coisa, seja eletrônico ou não) é um aspecto de circunstâncias físicas, de relevância para o comportamento do sistema” (COULOURIS *et al.*, 2013, p. 844). Em outras palavras, o contexto está relacionado ao mundo físico com o qual o usuário estará interagindo a partir de seu dispositivo móvel.

O reconhecimento do contexto se faz necessário pelo seguinte:

Como os usuários e os dispositivos que estamos considerando frequentemente são móveis, e como o mundo físico apresenta diferentes oportunidades de interações de locais em diferentes tempos, suas circunstâncias físicas são frequentemente relevantes como determinantes para o comportamento do sistema (COULOURIS *et al.*, 2013, p. 844).

Desta forma, a grande questão relacionada ao reconhecimento de contexto é que os dispositivos móveis não têm a capacidade de obter informações sobre o ambiente em que se encontram (contexto) quando fora de sua intranet base e adaptar-se automaticamente. O que ocorre é que cabe ao usuário fazer os ajustes de configurações manualmente à medida que se movem, algo considerado não desejável (ARAUJO, 2003 *apud* CIRILO, 2006).



"A mobilidade introduz vários desafios para os sistemas distribuídos, incluindo a necessidade de lidar com a conectividade variável e mesmo com a desconexão, e a necessidade de manter o funcionamento em face da mobilidade do aparelho" (COULOURIS et al., 2013, p. 10).

3 COMPUTAÇÃO PERVASIVA

Uma das principais características dos sistemas distribuídos sobre os quais vimos falando ao longo deste tópico estava relacionada ao fato de que eles eram constituídos de nós dispersos por uma estrutura de rede. Apesar de não sabermos exatamente qual nó da rede estaria executando algum processo, tínhamos a ideia de que este nó existia em algum lugar, de maneira fixa e com conexão permanente à rede (TANENBAUM; STEEN, 2007).

A introdução do conceito de computação móvel modificou este cenário favorecendo o surgimento de um tipo específico de sistemas distribuídos: os **sistemas pervasivos**. Neste tipo de sistema "[...] os equipamentos costumam ser caracterizados por seu pequeno tamanho, pela alimentação por bateria, por sua mobilidade e por terem somente uma conexão sem fio, se bem que nem todas essas características se aplicam a todos os dispositivos" (TANENBAUM; STEEN, 2007, p. 15).

Isso significa dizer que os sistemas pervasivos, ou computação pervasiva, estão à nossa volta e, por isso mesmo, caracterizam-se por ser desnecessário um controle administrativo por parte do usuário, ou seja, eles podem "controlar, configurar e ajustar a aplicação para melhor atender as necessidades do dispositivo ou usuário" (ARAÚJO, 2003 *apud* SANTAELLA, 2013, p. 17).

Essa capacidade dos dispositivos, sensores, entre outros, de descobrir o ambiente em que se encontram, e de outra forma, também o ambiente detectar outros dispositivos que venham a fazer parte dele, permitindo que se "encaixem", é resultante de uma interação inteligente entre eles (ARAÚJO, 2003 *apud* SANTAELLA, 2013; TANENBAUM; STEEN, 2007).



Perceba que, a partir dos sistemas pervasivos, os usuários de computação móvel podem se beneficiar de sistemas computacionais a partir de qualquer lugar.

Como é possível fazer com que um dispositivo se “encaixe” em um novo ambiente para o qual o usuário se deslocou? Basicamente, sistemas pervasivos precisam satisfazer três características que são (GRIMM *et al.*, 2004 *apud* TANENBAUM; STEEN, 2007):

- 1- Adotar mudanças contextuais:** falamos sobre reconhecimento de contexto quando estudamos as características da computação móvel. Então, em um ambiente de computação pervasiva o dispositivo deve “saber” que pode mudar de ambiente e se adequar a isso. Um exemplo simples é descobrir que uma rede não está mais disponível porque o usuário está se movimentando entre estações-base. Neste caso, o dispositivo deve se conectar à nova rede.
- 2- Incentivar composição *ad hoc*:** esta característica está diretamente relacionada ao comportamento das redes *ad hoc*. Este conceito foi introduzido a partir da proliferação das redes sem fio, e define o que chamamos de “rede espontânea”, ou seja, qualquer dispositivo (com fio ou sem fio) pode ser ligado a ela a qualquer momento. Assim, uma vez que, em ambientes de sistemas pervasivos, diferentes dispositivos podem ser usados por diferentes usuários de formas distintas, a configuração do conjunto de aplicações deve ser fácil.
- 3- Reconhecer compartilhamento como padrão:** considerando a natureza dos sistemas pervasivos, é natural o acesso e o compartilhamento de informações entre dispositivos. Considerando a possibilidade de mudança constante de dispositivos em um ambiente, é importante reconhecer que o espaço onde residem as informações mudará constantemente.

De maneira resumida, em se tratando de sistemas pervasivos, vale então lembrar o seguinte:

[...] na presença de mobilidade, dispositivos devem suportar a adaptação fácil e dependente de aplicação a seu ambiente local. Também devem ser capazes de descobrir serviços com eficiência e reagir de acordo. [...] na realidade, não existe transparência de distribuição em sistemas pervasivos. De fato, a distribuição de dados, processos e controle é *inerente* a esses sistemas, [...] (MASCOLO *et al.* 2004; NIEMELÄ; LTVAKOSKI, 2004 *apud* TANENBAUM; STEEN, 2007, p. 15, grifo do original).

4 COMPUTAÇÃO UBÍQUA

Agora que você compreendeu o conceito da computação móvel e da computação pervasiva, trataremos da computação ubíqua. A computação ubíqua foi representada anteriormente na Figura 30, como a intersecção entre a computação móvel e os sistemas pervasivos, ou seja, ela faz uso dos benefícios de ambos os conceitos.

Já comentamos anteriormente que na literatura é comum encontrarmos os termos ubíquo e pervasivo como sinônimos, mas agora que você já estudou sobre computação móvel e pervasiva ficará ainda mais claro para você que há diferenças entre os termos.

Perceba que, o termo pervasivo sugere que dispositivos de computação, geralmente de pequeno porte, farão cada vez mais parte do cotidiano das pessoas, a ponto de sequer serem notados. Por outro lado, o termo ubíquo “[...] dá a noção de que o acesso a serviços de computação está onipresente, isto é, disponível em qualquer lugar” (COULOURIS *et al.*, 2013, p. 10).

Historicamente, o termo “computação ubíqua” foi criado em 1988 por Mark Weiser, e significa “em toda parte”. “Weiser percebeu a predominância cada vez maior dos dispositivos de computação levando a mudanças revolucionárias na maneira como usariam os computadores” (COULOURIS *et al.*, 2013, p. 819).



Mark Weiser foi cientista chefe na Xerox PARC (Palo Alto Research Center).

A partir de seu ponto vista, a computação passaria por uma revolução em que uma pessoa teria muitos computadores. Todavia não da maneira convencional que temos hoje, na qual certamente você utiliza um computador desktop no trabalho e um notebook em casa, por exemplo. Na computação ubíqua, a ideia de uma pessoa para muitos computadores implica na existência de vários e diferentes dispositivos com capacidade computacional, com a finalidade de atender a diferentes tarefas (COULOURIS *et al.*, 2013). Veja como isso pode acontecer a partir do exemplo mostrado na Figura 33.

FIGURA 33 – COMPUTAÇÃO UBÍQUA



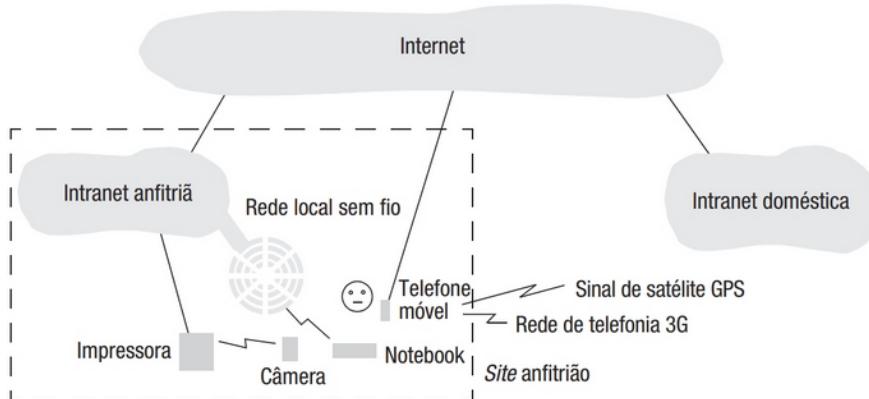
FONTE: <https://computacaoubiquaepervasiva.files.wordpress.com/2016/06/5814299287_d21359e54f_b.jpg?w=454&h=339>. Acesso em: 6 ago. 2019.

A segunda mudança que Weiser previu foi que os computadores “desapareceriam” – que eles “se incorporariam em utensílios e objetos do dia a dia, até se tornarem indistinguíveis”. Essa é principalmente uma noção psicológica, comparável ao modo como as pessoas aceitam normalmente a mobília de uma casa e mal a notam. Ela reflete a ideia de que a computação estará incorporada ao que consideramos itens do cotidiano – aqueles que normalmente não achamos que tenham recursos computacionais [...] (COULOURIS *et al.*, 2013, p. 820).

Pode-se dizer então que a computação ubíqua surge “da necessidade de se integrar mobilidade com a funcionalidade da computação pervasivas”, ou seja, qualquer dispositivo computacional, que levamos conosco, “pode construir dinamicamente modelos computacionais dos ambientes nos quais nos movemos e configurar seus serviços dependendo da necessidade” (ARAÚJO, 2003 *apud* SANTAELLA, 2013, p. 17).

De modo semelhante, “a computação ubíqua e a computação móvel se sobrepõem, pois, em princípio, o usuário móvel pode usar computadores que estejam em qualquer lugar. Porém, elas são distintas, de modo geral” (COULOURIS *et al.*, 2013, p. 10), como pode ser observado na Figura 34.

FIGURA 34 – DISPOSITIVOS MÓVEIS E PORTÁTEIS EM UM SISTEMA DISTRIBUÍDO



FONTE: COULOURIS *et al.* (2013, p. 11)

5 SEGURANÇA

Ao longo deste tópico abordamos a definição e características dos conceitos de computação móvel, ubíqua e pervasiva, como estão relacionadas e como cada uma acaba usufruindo dos benefícios das outras. Você percebeu também que o uso de dispositivos móveis e a possibilidade de se adaptar ao ambiente a partir de características como o reconhecimento de contextos e o compartilhamento de informações, tornam tais conceitos inerentemente distribuídos.

Perceba este contexto na afirmação de Weiser (1991 *apud* SILVA *et al.*, 2015, p. 28):

Com o desenvolvimento tecnológico das redes sem fio e a grande proliferação dos dispositivos portáteis, houve o aumento da popularidade da computação móvel, o que levou consequentemente, ao surgimento de aplicações e ambientes de computação ubíqua [...]. Neste sentido, surge a demanda por mecanismos de segurança que tornem sua aplicação confiável.

Repare que o autor frisa o surgimento de “demanda por mecanismos de segurança” em decorrência da popularização de dispositivos móveis e do consequente desenvolvimento de sistemas pervasivos e ubíquos.

Então, perceba que, em estruturas como estas há questões a serem consideradas em relação à segurança e, especialmente, privacidade. Lembre-se que você estudou sobre a importância da segurança quando falamos de sistemas distribuídos: para que se possa garantir os três pilares da segurança (confidencialidade, integridade e disponibilidade), é importante que se saiba com certeza a identidade do usuário/dispositivo envolvido no compartilhamento. E isto acaba sendo diminuído em ambientes ubíquos e/ou com o uso de dispositivos móveis (COULOURIS *et al.*, 2013).

Silva *et al.* (2015, p. 28) concordam com o pensamento de Couloris *et al.* (2013), declarando que:

Há necessidade de garantir serviços básicos de segurança confiáveis para redes sem fio para comodidade dos usuários. Para isso, considerase fundamental o apoio dos pilares da segurança da informação, tais como Confidencialidade, Integridade, Autenticidade e Disponibilidade. Com isso, impede-se falhas na troca de informações entre os dispositivos tais como interceptação ou modificação de seu conteúdo.

Repare que outro aspecto mencionado junto à segurança se refere à privacidade do usuário. A necessidade de implementação de mecanismos de segurança garante não somente os pilares relacionados à segurança da informação, mas permite ao usuário a possibilidade de o mesmo ser mantido no “anonimato”. Isso significa dizer que poderá haver momentos que o usuário pode não desejar ser localizado ou ter seus dados compartilhados (LEITHARDT *et al.*, 2014).

Nesse sentido, o compartilhamento de informações em ambientes pervasivos pode ser organizado de modo a reduzir processamento desnecessário a fim de aumentar os níveis de segurança e consequentemente o gerenciamento dos serviços disponíveis (LEITHARDT *et al.*, 2014).



[...] muitos usuários se preocupam com sua privacidade – grosso modo, sua capacidade de controlar a acessibilidade das informações sobre eles mesmos. Contudo, a privacidade está potencialmente mais ameaçada do que nunca, devido à percepção nos espaços inteligentes pelos quais os usuários passam (COULOURIS et al., 2013, p. 857).

Cabe ainda mencionar que, o aspecto relacionado à formação dos chamados “espaços inteligentes” ou *smartplaces* altera os ambientes físicos, que deixam de ser espaços exclusivos para circulação, habitação, lazer, entre outros, assumindo uma nova função de inserção de informações graças ao estabelecimento das redes de computação móvel. “As funções presentes nos dispositivos (especialmente aplicativos) passam a ser agentes potenciais de transformação no meio, inclusive com a transformação de relações territoriais de posse, pertencimento, localização e movimento [...]” (DALLABONA-FARINIUK; FIRMINO, 2018, p. 256).

Assim sendo, a constante mudança em relação à permanência ou não das pessoas em locais que oferecem ou não o acesso a uma rede Wi-Fi, alimenta a demanda por aplicativos que permitam o controle sobre o espaço e o reconhecimento das localidades (DALLABONA-FARINIUK; FIRMINO, 2018), ou ainda, pode-se também “[...] construir mecanismos de segurança através do auxílio dos *smartspaces*, os quais são gerenciados por um middleware, que tem por finalidade prover mecanismo de segurança” (SILVA et al., 2015, p. 28, grifo nosso).

Perceba que, apesar de se conhecerem as demandas relacionadas à segurança dos dados e privacidade do usuário em ambientes de computação móvel, ubíqua e pervasiva, ainda há uma lacuna a ser preenchida neste contexto.

Um aspecto especialmente crítico está relacionado a problemas com o hardware dos dispositivos: “a segurança e a privacidade são complicadas [...], por problemas relacionados ao hardware, como a escassez de recursos, e porque sua espontaneidade leva a novos tipos de compartilhamento de recursos” (COULOURIS et al., 2013, p. 858). Veja alguns destes problemas:

- “[...] dispositivos portáteis, como os *smartphones* [...], em geral podem ser mais facilmente roubados e falsificados do que os dispositivos como os PCs [...]” (ANDERSON et al., 2004 *apud* COULOURIS et al., 2013, p. 858), logo, ao considerar um projeto de segurança para um ambiente ubíquo, por exemplo, não se deve contar com a integridade de dispositivos que possam ser comprometidos.
- Eventualmente, os dispositivos não contam com recursos computacionais suficientes para suportar criptografia assimétrica, utilizando, normalmente, criptografia de chave simétrica por ser mais facilmente executável.

- O consumo de energia também se converte em um problema quando se fala de segurança. Neste sentido, os protocolos de segurança devem ser projetados para preservar o consumo da bateria do dispositivo.

No entanto, é importante ressaltar que, ao longo dos anos, vários são os esforços realizados para proporcionar a esse tipo de ambiente computacional o conceito de segurança, visando, em sua maioria, uma maneira de proporcionar aos nós da rede um meio de identificar e excluir nós maliciosos.



A Leitura Complementar selecionada para você nesta unidade aborda o conceito de "sistemas distribuídos de sistema" apresentado por Coulouris et al. (2013, p. 40).

LEITURA COMPLEMENTAR

SISTEMAS DISTRIBUÍDOS DE SISTEMAS

George Coulouris
Jean Dollimore
Tim Kindberg
Gordon Blair

Um relatório recente discute o surgimento de sistemas distribuídos USL (*Ultra Large Scale*) [www.sei.cmu.edu]. O relatório captura a complexidade dos sistemas distribuídos modernos, referindo-se a essas arquiteturas (físicas) como *sistemas de sistemas* (espelhando a visão da Internet como uma rede de redes). Um sistema de sistemas pode ser definido como um sistema complexo, consistindo em uma série de subsistemas, os quais são, eles próprios, sistemas que se reúnem para executar uma ou mais tarefas em particular.

Como exemplo de sistema de sistemas, considere um sistema de gerenciamento ambiental para previsão de enchentes. Nesse cenário, existirão redes de sensores implantadas para monitorar o estado de vários parâmetros ambientais relacionados a rios, terrenos propensos à inundação, efeitos das marés etc. Isso pode, então, ser acoplado a sistemas responsáveis por prever a probabilidade de enchentes, fazendo simulações (frequentemente complexas) em, por exemplo, *clusters computacionais* [...]. Outros sistemas podem ser estabelecidos para manter a analisar dados históricos ou para fornecer sistemas de alerta precoce para partes interessadas fundamentais, via telefones celulares.

FONTE: COULOURIS, G. et al. **Sistemas distribuídos**: conceitos e projetos. 5. ed. Porto Alegre: Bookman, 2013. Disponível em: <https://sdpisutic.files.wordpress.com/2017/08/sistemas-distribuc3addos-conceitos-e-projeto-2013.pdf>. Acesso em: 27 dez. 2018.

RESUMO DO TÓPICO 5

Neste tópico, você aprendeu que:

- Computação móvel, computação ubíqua e computação pervasiva são normalmente utilizadas como sinônimos, mas possuem diferenças conceituais importantes.
- Computação móvel define-se como a possibilidade de se movimentar levando junto consigo dispositivos que forneçam serviços computacionais.
- A computação móvel somente existe se os diferentes dispositivos se comunicam entre si a partir de conexões sem fio.
- Os dispositivos móveis apresentam como características: energia limitada, restrições de recurso, sensores e controladores.
- Computação pervasiva significa que o computador está embarcado no ambiente de forma invisível para o usuário.
- A computação pervasiva caracteriza-se por dispensar o controle administrativo do usuário, ou seja, os dispositivos devem ser capazes de descobrir e se encaixar no ambiente em que se encontram.
- Computação ubíqua implica na “[...] coordenação de dispositivos inteligentes, móveis e estacionários para prover aos usuários acesso imediato e universal à informação e novos serviços, de forma transparente” (ARAUJO, 2003 *apud* SANTAELLA, 2013, p. 16).
- A computação ubíqua se beneficia dos dois conceitos: computação móvel e computação pervasiva, já que implica na existência de vários dispositivos diferentes, com capacidade computacional, destinados a realizar diferentes tarefas.



Ficou alguma dúvida? Construímos uma trilha de aprendizagem pensando em facilitar sua compreensão. Acesse o QR Code, que levará ao AVA, e veja as novidades que preparamos para seu estudo.



AUTOATIVIDADE



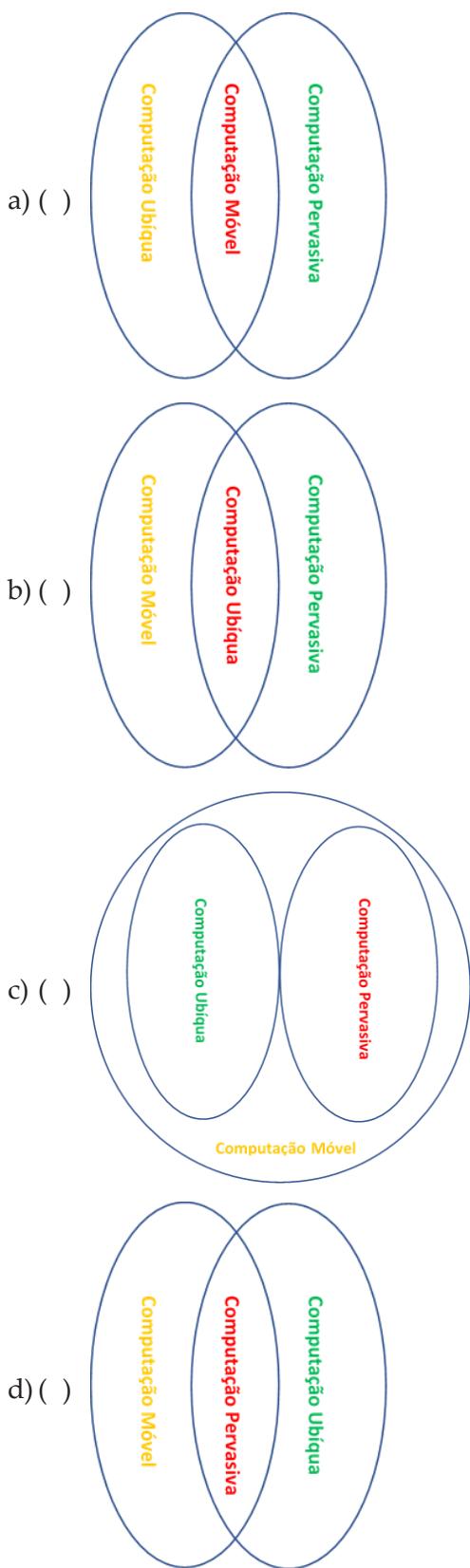
1 Após o nascimento e o amadurecimento da computação, com o advento das redes de computadores, o surgimento dos sistemas distribuídos e o nascimento da internet, culmina-se na miniaturização dos equipamentos, sua integração, e o surgimento de um novo conceito de computação, a computação móvel e a computação ubíqua, e orbitando nesse ambiente a computação pervasiva. Sobre esses novos paradigmas avalie as sentenças:

- () Quando um usuário, levando consigo o seu celular, *tablet*, *notebook*, entre outros, está conectado a uma rede A e muda para uma rede B, e com isso leva com ele os serviços computacionais que usava na rede A, e consequentemente caracterizando que esses serviços tornam-se um elemento que pode ser movimentado entre redes a qualquer momento caracteriza o conceito fundamental da computação ubíqua.
- () Quando um usuário, levando consigo o seu celular, *tablet*, *notebook*, entre outros, está conectado a uma rede A e muda para uma rede B, e com isso leva com ele os serviços computacionais que usava na rede A, e consequentemente caracterizando que esses serviços tornam-se um elemento que pode ser movimentado entre redes a qualquer momento, caracteriza o conceito fundamental da computação móvel.
- () Quando um usuário, levando consigo o seu celular, *tablet*, *notebook*, entre outros, está conectado a uma rede A e muda para uma rede B, e com isso leva com ele os serviços computacionais que usava na rede A, e consequentemente caracterizando que esses serviços tornam-se um elemento que pode ser movimentado entre redes a qualquer momento caracteriza o conceito fundamental da computação pervasiva.
- () Imagine um ambiente de serviços computacionais (acesso a impressoras, serviços Wi-Fi, compartilhamento de mídias, entre outros) e que o mesmo seja totalmente transparente para quem o utiliza. Nesse ambiente, os dispositivos têm a capacidade de extrair informações desse ambiente e as utilizar na criação de novos modelos computacionais, isso caracteriza o conceito fundamental da computação pervasiva.

Agora, assinale a alternativa CORRETA:

- a) () V – V – V – V.
- b) () V – F – V – F.
- c) () V – F – F – V.
- d) () F – V – F – V.

- 2 Onovo ambiente criado pela computação móvel, ubíqua e pervasiva lida, muitas vezes, com dispositivos que possuem recursos limitados (principalmente processamento e memória). Um outro aspecto relativo a essas limitações lida com o consumo de energia desses dispositivos. Com isso em mente, assinale a alternativa CORRETA sobre a relação entre esses paradigmas:



- 3 O novo ambiente criado pela computação móvel, ubíqua e pervasiva lida, muitas vezes, com dispositivos que possuem recursos limitados (principalmente processamento e memória). Um outro aspecto relativo a essas limitações lida com o consumo de energia desses dispositivos. Com isso em mente, avalie as sentenças sobre as características desses dispositivos:
- I- O recurso para o consumo de energia dos dispositivos móveis é inversamente proporcional às suas velocidades de processamento e às suas capacidades de armazenamento. Dessa forma, uma melhoria no tocante ao consumo de energia não implicaria diretamente em um maior poder computacional desses dispositivos.
 - II- O recurso para o consumo de energia dos dispositivos móveis é diretamente proporcional às suas velocidades de processamento e às suas capacidades de armazenamento. Dessa forma, uma melhoria no tocante ao consumo de energia implicaria diretamente em um maior poder computacional desses dispositivos.
 - III- O ambiente da computação móvel trouxe à tona dispositivos como controladores e sensores. Os controladores respondem pela medição de valores físicos para fornecê-los a algum software (Sistema de Posicionamento Global, por exemplo), já os sensores obedecem aos softwares, afetando dessa forma o ambiente físico no qual se encontram.
 - IV- O ambiente da computação móvel trouxe à tona dispositivos como controladores e sensores. Os sensores respondem pela medição de valores físicos para fornecê-los a algum software (Sistema de Posicionamento Global, por exemplo), já os controladores obedecem aos softwares, afetando dessa forma o ambiente físico no qual se encontram.

Agora, assinale a alternativa CORRETA:

- a) () As sentenças I e II estão corretas.
- b) () As sentenças III e IV estão corretas.
- c) () A sentença III está incorreta e a sentença IV está correta.
- d) () A sentença I está correta e a sentença III está incorreta.

- 4 Na computação pervasiva deve existir uma propriedade inerente aos dispositivos (sensores, controladores e outros) de ter consciência do ambiente ao qual pertencem, e por outro lado, esse ambiente precisa perceber tais dispositivos integrantes dele, isso garante que eles se integrem através de interações inteligentes entre si. Dessa forma, assinale a afirmação CORRETA sobre as características desse tipo de sistema:

- a) () A característica de um dispositivo pervasivo perceber que uma conexão não está mais disponível devido a um movimento desse dispositivo entre redes é conhecida como composição *ad hoc*.
- b) () A característica de um dispositivo pervasivo perceber que uma conexão não está mais disponível devido a um movimento desse dispositivo entre redes é conhecida como reconhecimento de compartilhamento como padrão.

- c) () A característica de um dispositivo pervasivo perceber que uma conexão não está mais disponível devido a um movimento desse dispositivo entre redes é conhecida como adoção de mudanças contextuais. Sendo que, a natureza desses sistemas não permite acesso e compartilhamento de informações entre os dispositivos conectados.
- d) () A característica de um dispositivo pervasivo perceber que uma conexão não está mais disponível devido a um movimento desse dispositivo entre redes é conhecida como adoção de mudanças contextuais.
- 5 Sabe-se que o novo ambiente criado pela computação móvel, ubíqua e pervasiva lida, muitas vezes, com dispositivos que possuem recursos limitados (principalmente processamento e memória). Sendo assim, nesse tipo de ambiente deve-se levar em consideração questões ligadas a segurança, principalmente a privacidade. Lembre-se que o tripé que fundamenta a segurança nos sistemas distribuídos é baseado na confidencialidade, integridade e disponibilidade. Sabendo disso, avalie as sentenças sobre os problemas de segurança em sistemas distribuídos, classificando V para as sentenças verdadeiras e F para as falsas:
- () A característica portátil dos dispositivos distribuídos (smartphones, sensores, controladores, tablets, entre outros) não representa um problema, pois não são facilmente roubados ou falsificados.
- () O item recurso computacional dos dispositivos distribuídos não é um fator relevante para ser considerado na construção de sistemas distribuídos, já que esses dispositivos podem ter embarcados em sua estrutura interna desde recursos para processamento de dados até aqueles que envolvem criptografia.
- () Consumir energia é um fator inerente a todo dispositivo eletrônico, seja ele distribuído ou não. Porém quando ele está associado ao processamento e armazenamento de dispositivos executando funções em ambientes distribuídos, converte-se em um item extremamente relevante e complexo.
- () Consumir energia é um fator inerente a todo dispositivo eletrônico, seja ele distribuído ou não. Porém quando ele está associado ao processamento e armazenamento de dispositivos executando funções em ambientes distribuídos, converte-se em um item extremamente relevante, porém simples de solucionar.

Agora, assinale a alternativa CORRETA:

- a) () F – F – V – F.
- b) () V – V – F – V.
- c) () V – F – V – F.
- d) () F – V – F – V.

COMUNICAÇÃO EM AMBIENTES DISTRIBUÍDOS E APLICAÇÕES DISTRIBUÍDAS

OBJETIVOS DE APRENDIZAGEM

A partir do estudo desta unidade, você será capaz de:

- compreender como acontece a comunicação entre dispositivos em sistemas distribuídos;
- entender a importância dos protocolos de rede para comunicação entre esses dispositivos e o processo envolvendo requisições e respostas em ambientes distribuídos;
- compreender a importância relacionada à replicação de dados em sistemas distribuídos e sua relação com outros tópicos de interesse como a consistência, disponibilidade e tolerância a falhas;
- aprender sobre segurança, suas visões, além de seus mecanismos e técnicas aplicados aos sistemas distribuídos;
- conhecer aplicações práticas de sistemas distribuídos, suas arquiteturas e processos e como lidam com os aspectos de consistência, tolerância a falhas e segurança.

PLANO DE ESTUDOS

Esta unidade está dividida em quatro tópicos. No decorrer da unidade você encontrará autoatividades com o objetivo de reforçar o conteúdo apresentado.

TÓPICO 1 – COMUNICAÇÃO

TÓPICO 2 – CONSISTÊNCIA E TOLERÂNCIA A FALHAS

TÓPICO 3 – SEGURANÇA

TÓPICO 4 – IMPLEMENTAÇÕES DE SISTEMAS E APLICAÇÕES DISTRIBUÍDAS



Preparado para ampliar seus conhecimentos? Respire e vamos em frente! Procure um ambiente que facilite a concentração, assim absorverá melhor as informações.

COMUNICAÇÃO

1 INTRODUÇÃO

Você deve recordar que, na Unidade 2, tratamos dos diferentes tipos de processos e do seu importante papel nos sistemas distribuídos. Conceitos como os de *thread* e de virtualização foram introduzidos, além de uma importante questão relacionada aos sistemas distribuídos em longas distâncias que é a movimentação de processos entre máquinas diferentes, ou seja, a migração de código.

Mantendo como foco principal os processos, nos dedicaremos agora a tratar sobre como ocorre a comunicação entre os processos em sistemas distribuídos. Conforme já comentamos várias vezes ao longo deste livro, os sistemas distribuídos fazem uso das estruturas de redes de computadores (sejam elas locais ou de longa distância) para se comunicarem. Logo, uma série de características relacionadas às redes de computadores afeta o projeto e o comportamento dos sistemas distribuídos e sua comunicação (COULOURIS *et al.*, 2013).

Sendo assim, é imprescindível que você compreenda a importância da comunicação nos sistemas distribuídos:

Comunicação entre processos está no coração de todo sistema distribuído. Não tem sentido estudar sistemas distribuídos sem examinar cuidadosamente os modos pelos quais processos em máquinas diferentes podem trocar informações. A comunicação em sistemas distribuídos é sempre baseada em troca de mensagens de baixo nível como a oferecida pela rede subjacente (TANENBAUM; VAN STEEN, 2007, p. 69).



O termo “rede subjacente” utilizado pelo autor refere-se a uma das redes sobre a qual o sistema distribuído está sendo executado. Lembre-se do que vimos falando ao longo do livro: sistemas distribuídos funcionam sobre estruturas de redes de computadores.

O projeto de desenvolvimento de sistemas distribuídos, e, consequentemente, seu esquema de comunicação torna-se complexo por uma série de razões que envolvem desde as características do SO instalado em cada computador, até as características das redes que os interligam. Isso significa dizer que um sistema distribuído trabalha com um sem número (provavelmente na casa dos milhares) de processos espalhados por uma rede, cujos recursos de comunicação nem sempre são confiáveis, fato que não deve ser desprezado e que impacta diretamente na complexidade para projetos de sistemas distribuídos (COULOURIS *et al.*, 2013; TANENBAUM; VAN STEEN, 2007).

Por exemplo, para que você tenha ideia da complexidade sobre a qual acabamos de falar, vamos considerar a internet. “A Internet é um subsistema de comunicação único que fornece comunicação entre todos os hosts conectados a ela. Ela é construída a partir de muitas sub-redes” (COULOURIS *et al.*, 2013, p. 82), o que inclui um conjunto heterogêneo de componentes, envolvendo mídias de transmissão, hardware e software (protocolos, rotinas de tratamento de comunicação). Assim, a infraestrutura da internet deve ser capaz de integrar toda esta variedade de componentes em um único serviço de comunicação (COULOURIS *et al.*, 2013).

Perceba então que, para tratarmos da comunicação entre processos em um sistema distribuído, é importante que façamos um exercício de recapitação de alguns conceitos essenciais, envolvendo as redes de computadores e seus protocolos, estes últimos essenciais para o entendimento da comunicação entre processos distribuídos. Esses temas serão tratados na sequência.

2 REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS

Como você já aprendeu, as redes de computadores são de grande importância para o funcionamento de sistemas distribuídos. Isso porque é através de sua infraestrutura, organização e implementação de seus protocolos que ocorre a comunicação entre os processos nos sistemas distribuídos.

Os diferentes tipos de redes de computadores, desde as redes pessoais até as redes de longa distância, estabelecem não somente sua organização física, mas também (e mais importante para este nosso estudo), quais são as tecnologias empregadas na transmissão de dados.

O Quadro 1 exemplifica alguns dos principais tipos de redes de computadores, diferenciando as estruturas que utilizam cabo como meio de comunicação daquelas sem fio (COULOURIS *et al.*, 2013).

QUADRO 1 – CARACTERÍSTICAS DE DESEMPENHO DOS VÁRIOS TIPOS DE REDES DE COMPUTADORES

Tipo de Rede	Exemplo	Alcance	Largura de banda (Mbps)	Latência (ms)
Redes cabeadas				
LAN	Ethernet	1 - 2 km	10 - 10.000	1 - 10
WAN	Roteamento IP	mundial	0,010 - 600	100 - 500
MAN	ATM	2 - 50 km	1 - 600	10
Interligação em rede	Internet	mundial	0,5 - 600	100 - 500
Redes sem fio				
WPAN	Bluetooth (IEEE 802.15.1)	10 - 30 m	0,5 - 2	5 - 20
WLAN	Wi-Fi (IEEE 802.11)	0,15 - 1,5 km	11 - 108	5 - 20
WMAN	WiMAX (IEEE 802.16)	5 - 50 km	1,5 - 20	5 - 20
WWAN	Redes telefônicas 3G e 4G	celular: 1 - 5 km	348 - 14,4	100 - 500

FONTE: Adaptado de Coulouris et al. (2013, p. 86)

- **Redes locais (LAN – Local Area Networks):** são assim chamadas por cobrirem pequenas distâncias, como um prédio de escritórios ou um campus universitário, por exemplo, a partir de uma infraestrutura cabeada. LAN são consideradas redes de alto desempenho pois apresentam velocidade de comunicação relativamente alta e baixa latência. Várias são as tecnologias para a implementação deste tipo de rede, sendo a Ethernet a tecnologia dominante.
- **Redes metropolitanas (MAN – Metropolitan Area Networks):** tipo específico de rede que faz uso de cabos de fibra ótica ou cobre com uma cobertura de até 50 quilômetros, aproximadamente, dentro de uma cidade. Por possuírem considerável largura de banda, são utilizadas para transmitir vídeo e voz.
- **Redes de longa distância (WAN – Wide Area Networks):** mais lentas que as redes locais, as WAN permitem a comunicação entre *hosts* que se encontram em cidades, países e até continentes diferentes. Nas WAN a transmissão de mensagens é gerenciada a partir de roteadores.



"[...] as operações de roteamento introduzem um atraso em cada um dos pontos de uma rota; portanto, a latência total da transmissão de uma mensagem depende da rota que ela segue e das cargas de tráfego nos diversos segmentos de rede pelos quais ela passa" (COULOURIS et al., 2013, p. 87).

- **Redes pessoais sem fio (WPAN – Wireless Personal Area Networks):** consideradas uma subcategoria das redes locais, vem ganhando cada vez mais destaque dada a popularização de dispositivos pessoais móveis transportados pelo usuário, que podem ser conectados em redes de baixos custo e energia.
- **Redes locais sem fio (WLAN – Wireless Local Area Networks):** a intenção de uma LAN sem fio é permitir a conexão entre dispositivos móveis dentro de prédios (tal qual como as LAN cabeadas), eliminando a necessidade de fios.
- **Redes metropolitanas sem fio (WMAN – Wireless Metropolitan Area Network):** “o padrão IEEE 802.16 WiMAX é destinado a essa classe de rede. Seu propósito é fornecer uma alternativa às conexões cabeadas para casas ou prédios de escritório e substituir as redes 802.11 WiFi em algumas aplicações” (COULOURIS *et al.*, 2013, p. 88).
- **Redes de longa distância sem fio (WWAN – Wireless Wide Area Networks):** o principal exemplo desta categoria de rede são as redes de telefonia móvel. Para que possam operar em áreas de longa distância (como em um país inteiro), as redes de telefonia móvel utilizam conexões de rádio, cujo sinal é confinado em regiões denominadas células (daí o nome celular). O alcance, ou cobertura, ocorre pela interligação e superposição dessas células.

Finalmente, é necessário sabermos que a interligação em redes é importante para o desenvolvimento de sistemas distribuídos. Uma interligação em rede trata-se de “[...] um subsistema de comunicação em que várias redes são unidas para fornecer recursos de comunicação de dados comuns, abstraindo as tecnologias e os protocolos das redes componentes individuais [...]” (COULOURIS *et al.*, 2013, p. 88).

2.1 PROTOCOLOS DE COMUNICAÇÃO

Agora que você recordou como as redes de computadores podem ser organizadas, tornou-se ainda mais claro que seu requisito fundamental é a transmissão de informações entre os vários nodos que a compõem. Importante também lembrar que a transmissão de informações pelas redes é mais conhecida como transmissão de mensagens. Para que uma mensagem seja transmitida em uma estrutura de rede, normalmente ela é dividida em pacotes, os quais são transmitidos pelo nodo emissor, e recebidos, corretamente, em seu destino (nodo receptor).



“Pacote’ é um termo genérico para referenciar uma sequência de dados binários com tamanho limitado usado como unidade de transmissão. Entretanto, conceitualmente, cada camada de um protocolo de rede define sua própria unidade de transmissão [...]” (COULOURIS *et al.*, 2013, p. 89).

Você poderia então se perguntar: “considerando uma estrutura de rede como a internet, que está em constante crescimento, como seria possível garantir que um pacote enviado por um sistema aqui no Brasil possa chegar corretamente ao seu destino em outro lugar no mundo?”. A resposta a essa pergunta está no que chamamos (e você já deve lembrar!) de protocolos de comunicação.

Por definição, um protocolo de comunicação representa um conjunto de regras bem formalizadas que estabelecem o formato, conteúdo e significado das mensagens enviadas e recebidas entre processos (COULOURIS *et al.*, 2013; TANENBAUM; VAN STEEN, 2007). Em outras palavras, é por causa da existência dessas regras que as mensagens entre diferentes sistemas, a partir de diferentes dispositivos, podem ser transferidas corretamente.

Perceba então que existem dois aspectos imprescindíveis para nossa compreensão sobre o que são protocolos e sua importância nos sistemas de comunicação: (1) especificam o formato dos dados nas mensagens (garantia de que a mensagem enviada é exatamente a mesma que será recebida); e (2) especificam a sequência das mensagens trocadas (COULOURIS *et al.*, 2013).

A importância dos protocolos para sistemas distribuídos é então estabelecida à medida que percebemos o seguinte:

A existência de protocolos bem-conhecidos permite que os vários componentes de software de um sistema distribuído sejam desenvolvidos e implementados de forma independente, usando diferentes linguagens de programação, em computadores que podem usar distintas representações internas para códigos e dados (COULOURIS *et al.*, 2013, p. 92).

Uma vez que você já compreendeu o que são os protocolos, podemos agora então passar para um próximo assunto de nosso interesse, que trata de como os protocolos são organizados. Como já comentamos anteriormente, toda a comunicação em sistemas distribuídos é baseada na troca de mensagens. E como também já comentamos, são os protocolos que garantem que esta comunicação se dará de forma adequada (de maneira correta e na exata sequência em que as mensagens foram enviadas) (TANENBAUM; VAN STEEN, 2007).

Exatamente porque estamos falando de um conjunto de ações muito complexas, para as quais são necessários acordos em vários níveis, desde o nível mais baixo que trata da transmissão de bits, até um nível mais alto que estabelece como a informação deverá ser expressa (TANENBAUM; VAN STEEN, 2007), os protocolos de comunicação foram organizados utilizando uma estrutura em camadas, também chamada de níveis ou hierarquia.



"Um conjunto completo de camadas de protocolo é referido como conjunto de protocolos ou pilha de protocolos, refletindo a estrutura em camadas" (COULOURIS et al., 2013, p. 94).

Em uma estrutura em camadas, cada camada "[...] é representada por um módulo de software em cada computador conectado à rede" (COULOURIS et al., 2013, p. 93). Assim cada camada é devidamente identificada e possui uma interface para as camadas que estão acima e/ou abaixo dela. Desta forma, é possível saber qual nível deve fazer qual serviço.

Veja, na Figura 1, uma ilustração de estrutura em camadas de protocolos. Perceba que para uma mensagem ser enviada da origem ao seu destinatário ela percorre, verticalmente, as camadas de protocolos até chegar ao nível mais baixo, que então encaminha a mensagem ao destinatário. No destinatário a mensagem percorre o caminho inverso, sendo recebida pela camada mais baixa, seguindo para as camadas superiores até a camada mais alta, que poderá finalmente apresentar a mensagem para o usuário no destino. A comunicação entre as camadas acima e abaixo se dá a partir de chamadas de procedimentos locais.

FIGURA 1 – ORGANIZAÇÃO CONCEITUAL DOS PROTOCOLOS EM CAMADAS



FONTE: Couloris et al. (2013, p. 92)

Visualizando assim a forma como acontece a comunicação entre dois sistemas distintos fica mais fácil entender por que comentamos antes que se trata de “um conjunto de ações muito complexas” não é mesmo? Assim, para tornar mais fácil todas as questões e níveis envolvidos na comunicação, a ISO (*International Organization for Standardization*) desenvolveu um modelo de referência com sete camadas para interconexão de sistemas abertos, o modelo OSI (*Open System Interconnection*) (COULOURIS *et al.*, 2013; TANENBAUM; VAN STEEN, 2007).

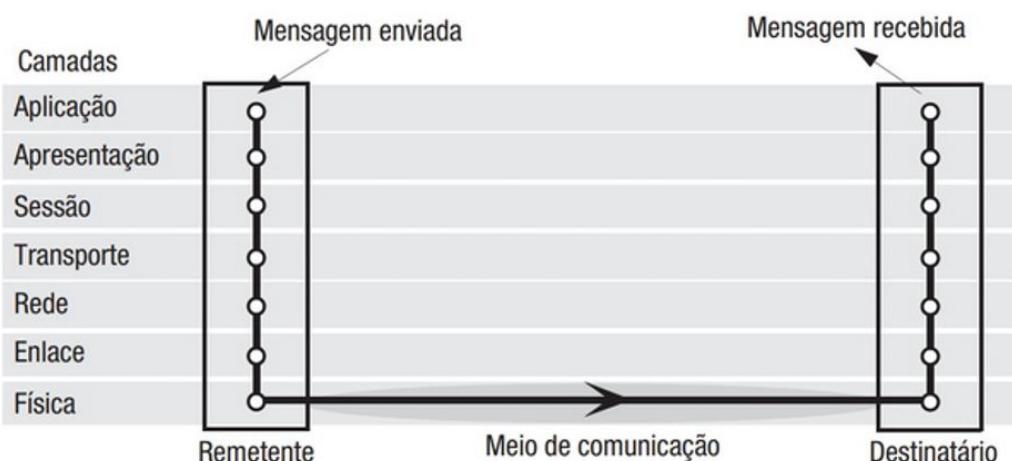
Como o próprio nome sugere, o modelo OSI foi criado para permitir o desenvolvimento de padrões de protocolos para comunicação entre sistemas abertos. “Um sistema aberto é o que está preparado para se comunicar com qualquer outro sistema aberto, usando regras padronizadas que regem o formato, o conteúdo e o significado das mensagens recebidas” (TANENBAUM; VAN STEEN, 2007, p. 70).



“[...] os protocolos que foram desenvolvidos como parte do modelo OSI nunca foram amplamente utilizados e, hoje em dia, estão definitivamente mortos e enterrados. Contudo, o modelo [...] em si mostrou ser bastante útil para entender redes de computadores” (TANENBAUM; VAN STEEN, 2007, p. 70).

A Figura 2 mostra a pilha de protocolos do modelo de referência OSI

FIGURA 2 – CONJUNTO DE CAMADAS DO MODELO OSI



FONTE: Coulouris *et al.* (2013, p. 94)

2.2 ATRIBUTOS DAS REDES PARA OS SISTEMAS DISTRIBUÍDOS

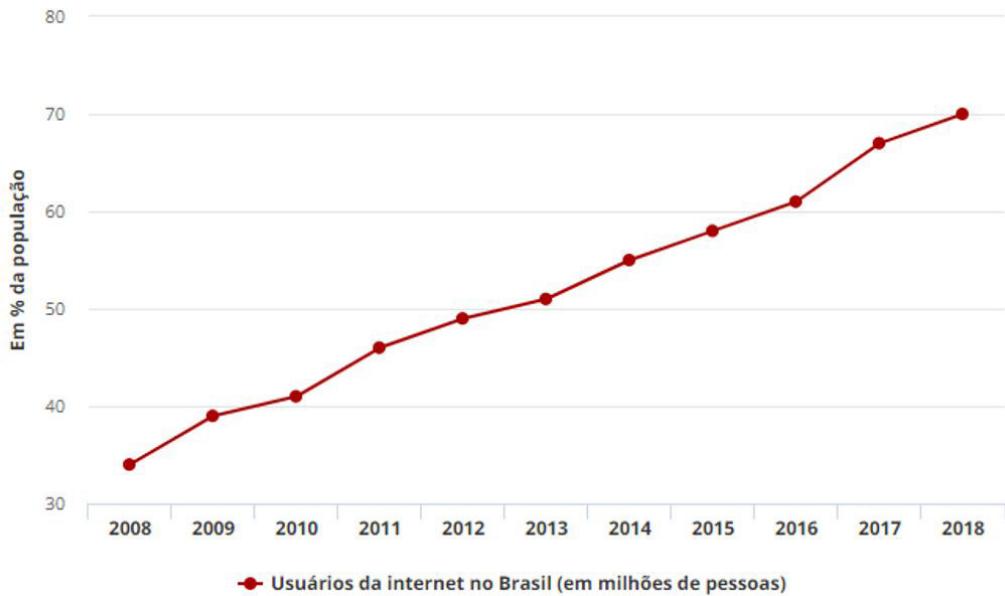
Agora que você já estudou sobre a organização das redes de computadores e como seus protocolos são imprescindíveis para o suporte à troca de mensagens entre os diferentes dispositivos que compõem os sistemas distribuídos, você será convidado a conhecer certas características necessárias às redes, de modo que permitam o funcionamento adequado dos sistemas distribuídos.

A evolução tecnológica que você viu representada na Unidade 1 não se limitou somente aos computadores e a seu SO. Também as redes de computadores, inicialmente projetadas para realizar funcionalidades relativamente simples, como o envio de e-mails, evoluíram para suportar aplicações cada vez mais complexas. Exemplo disso são os sistemas distribuídos que exigiram, como você já estudou, o atendimento de requisitos como acesso a arquivos compartilhados e outros recursos de software e hardware.

Atualmente, outros requisitos precisam ser atendidos para suportar a complexidade da comunicação em sistemas distribuídos (COULOURIS *et al.*, 2013; DEITEL; DEITEL; CHOHNES, 2005).

- **Desempenho:** diferentemente de sistemas centralizados nos quais as requisições de todos os usuários são tratadas por um único servidor, em sistemas distribuídos “[...] as requisições de usuários podem ser enviadas a diferentes servidores que trabalham em paralelo para aumentar o desempenho” (DEITEL; DEITEL; CHOHNES, 2005, p. 507). O desempenho em uma rede está relacionado à medição de alguns parâmetros que afetam a velocidade de transferência de mensagens entre dois computadores. Esses parâmetros envolvem (1) latência (entendida como o tempo decorrido entre o envio e o recebimento de uma mensagem, é determinada pelas características físicas da rede); e (2) taxa de transferência (velocidade na qual os dados são transmitidos entre dois computadores, está relacionada a sobrecargas de software).
- **Escalabilidade:** este atributo “[...] permite que um sistema distribuído cresça (adicone mais máquinas ao sistema) sem afetar as aplicações e os usuários existentes” (DEITEL; DEITEL; CHOHNES, 2005, p. 507). Desde seu surgimento as redes de computadores vêm sofrendo um crescimento rápido e diversificado (justificado pelos tipos existentes que você acabou de estudar). O crescimento alcançado pela internet nas últimas décadas, tanto em número de *hosts* como de usuários, conforme você pode observar na Figura 3, tende a indicar que seu tamanho futuro será proporcional à população do planeta, chegando à casa dos bilhões. Assim, é importante reconhecer que as atuais tecnologias de rede não foram projetadas para suportar a escala atual da rede, e, por isso, já são consideradas mudanças nos atuais mecanismos de endereçamento e roteamento.

FIGURA 3 – USUÁRIOS DE INTERNET NO BRASIL



FONTE: Adaptado de TIC domicílios (2018 *apud* LAVADO, 2019)



Até 2012 o maior número de servidores de internet se encontrava nos Estados Unidos (505.000.000), seguido do Japão (64.453.000) e do Brasil (26.577.000). Já o país que registrou o maior número de usuários de internet até 2016 foi a China, com 730.723.968, seguido da Índia com 374.328.160 usuários e Estados Unidos, com 246.809.216. Em quantidade de usuários o Brasil aparece na quarta posição, logo depois dos Estados Unidos, com 122.841.216 usuários (INDEX MUNDI, 2018a; 2018b).

- **Confiabilidade:** lembre-se que confiabilidade em sistemas distribuídos é um requisito diretamente relacionado ao impacto causado por eventuais falhas de comunicação e pela capacidade dos aplicativos em se recuperar quando tais falhas acontecem. Assim, “a confiabilidade da maior parte da mídia de transmissão física é muito alta. Quando ocorrem erros, é normalmente devido a falhas no software presente no remetente ou no destinatário [...], em vez de erros na rede” (COULOURIS *et al.*, 2013, p. 84).
- **Segurança:** a criação de medidas de segurança para sistemas distribuídos está relacionada à garantia de privacidade, integridade e disponibilidade dos recursos que os compõem. Em outras palavras, é preciso que se assegure a segurança, já que ao se falar em sistemas distribuídos, naturalmente está se falando em compartilhamento de recursos. Para tanto, as empresas costumam adotar políticas de segurança, postas em prática através de mecanismos de segurança.



É importante que você saiba a diferença entre políticas de segurança e mecanismos de segurança, por isso mesmo trataremos desse assunto com detalhes mais à frente, no Tópico 3. Por enquanto, basta sabermos que uma política de segurança descreve quais ações podem ser realizadas por algum recurso do sistema, enquanto os mecanismos de segurança são as ações aplicáveis para que uma política seja imposta (TANENBAUM; VAN STEEN, 2007).

- **Mobilidade:** a grande questão relacionada à mobilidade em sistemas distribuídos está no fato de que dispositivos móveis, como o seu smartphone, por exemplo, mudam de lugar e são reconectados a diferentes pontos de rede frequentemente. “Os mecanismos da Internet foram adaptados e ampliados para suportar mobilidade, mas o crescimento esperado para o uso de dispositivos móveis exigirá ainda mais desenvolvimentos” (COULOURIS *et al.*, 2013, p. 85).
- **Difusão seletiva (*multicasting*):** “a maior parte da comunicação em sistemas distribuídos se dá entre pares de processos, mas frequentemente também há necessidade de uma comunicação de um para muitos” (COULOURIS *et al.*, 2013, p. 85). Fazendo com que tecnologias de rede forneçam suporte ao envio de mensagens para vários destinatários.

3 CARACTERÍSTICAS GERAIS DA COMUNICAÇÃO ENTRE PROCESSOS EM SISTEMAS DISTRIBUÍDOS

Conforme você já estudou anteriormente, o projeto e desenvolvimento de sistemas distribuídos é uma tarefa bastante complexa, convertendo-se em um grande desafio para os projetistas. Isso porque sistemas distribuídos são compostos por uma grande quantidade de processos (na casa dos milhares, até milhões) espalhados por redes de computadores, como a internet, por exemplo (DEITEL; DEITEL; CHOFFNES, 2005; TANENBAUM; VAN STEEN, 2007), dificultando o gerenciamento da comunicação entre os dispositivos.

Desta forma, sistemas distribuídos heterogêneos exigem que se garanta a interoperabilidade entre aplicações.

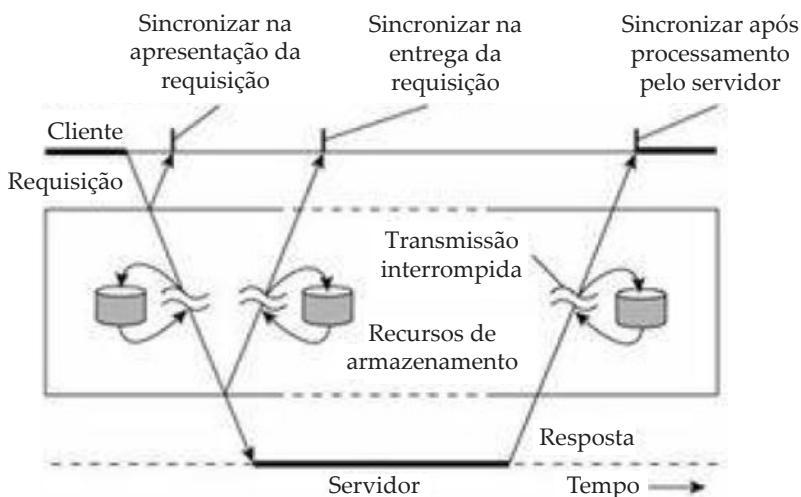
Interoperabilidade permite que componentes de software interajam entre hardwares e plataformas de software diferentes, linguagens de programação e protocolos de comunicação distintos. Uma interface permite que sistemas heterogêneos se comuniquem de modo significativo para ambos os lados — promovendo a interoperabilidade. No modelo cliente/servidor, o cliente emite uma requisição ao servidor através da rede e este processa a requisição e envia a resposta de

volta ao computador cliente, frequentemente usando uma interface para auxiliar a comunicação. Uma interface padronizada permite que cada par cliente/servidor se comunique utilizando uma interface única, comum, entendida por ambos os lados (DEITEL; DEITEL; CHOIFFNES, 2005, p. 510).

Conforme já estudamos na Unidade 2, por serem implementados como uma camada intermediária entre os SO locais e as aplicações utilizadas pelos usuários, os sistemas distribuídos são também denominados *middleware*. O *middleware* contém uma série de protocolos de comunicação de alto nível e protocolos capazes de estabelecer vários serviços, promovendo então esta interoperabilidade (TANENBAUM; VAN STEEN, 2007).

Então, para que você possa compreender as várias alternativas de comunicação que o *middleware* pode oferecer, é possível mostrá-lo como um “[...] serviço adicional de computação cliente/servidor” (TANENBAUM; VAN STEEN, 2007, p. 74) conforme a Figura 4.

FIGURA 4 – MIDDLEWARE COMO UM SERVIÇO NA COMUNICAÇÃO EM NÍVEL DE APLICAÇÃO CLIENTE/SERVIDOR



FONTE: Tanenbaum e Van Steen (2007, p. 74)

O esquema apresentado na Figura 4 pode ser mais bem compreendido se você o comparar ao envio e recebimento de um e-mail. De maneira geral, neste tipo de serviço, um cliente identificado como remetente (que poderia ser você) escreve uma mensagem e passa para o sistema de entrega do “correio” no servidor, e espera que a mensagem seja entregue ao seu destino (que poderia ser um professor na UNIASSELVI). Do outro lado, o receptor se conecta ao serviço de “correio” para verificar se alguma mensagem chegou.

Esta troca de mensagens entre dois processos é suportada por duas operações denominadas *send* e *receive*, respectivamente envio e recebimento, e sua execução implica em outro aspecto também ilustrado na Figura 4, denominado sincronização (COULOURIS *et al.*, 2013).

A comunicação entre os processos origem e destino pode ser síncrona ou assíncrona (COULOURIS *et al.*, 2013; TANENBAUM; VAN STEEN, 2007).

- **Comunicação síncrona:** neste tipo de comunicação os processos de origem e de destino são sincronizados a cada envio/recebimento de mensagens. Isso ocorre porque as operações *send* e *receive* causam bloqueio; na prática, o remetente é bloqueado até saber que sua requisição foi aceita.
- **Comunicação assíncrona:** neste caso o remetente de uma mensagem continua sua execução imediatamente após apresentar sua mensagem para transmissão, o que significa dizer que a operação *send* não implica em bloqueio. Isso é possível porque a mensagem é copiada e armazenada em um *buffer* local pelo *middleware*, permitindo que a transmissão da mensagem ocorra em paralelo ao processo de origem. Por outro lado, na comunicação assíncrona a operação *receive* pode ou não sofrer bloqueio. No caso de não sofrer bloqueio, o processo destino continua em execução após a realização do *receive*, que preenche um *buffer* com os dados a serem lidos.

3.1 CHAMADA DE PROCEDIMENTO REMOTO

Como você estudou até agora, de maneira geral, sistemas distribuídos baseiam-se na troca de mensagens (pacotes) entre processos, e, para tanto, fazem uso de duas operações básicas: *send* e *receive*. Contudo, em se tratando de sistemas distribuídos, é importante que haja transparência na comunicação, ou seja, importante que se possa manter a comunicação com um dispositivo localizado remotamente como se ele fosse local, e essa transparência não é possível com essas duas operações básicas (TANENBAUM; VAN STEEN, 2007).

Para resolver este problema, em meados da década de 1980 foi proposto um modelo diferente para a realização da comunicação em sistemas distribuídos, estabelecendo-se então o conceito de chamada de procedimento remoto (RPC) (DEITEL; DEITEL; CHOHNES, 2005; TANENBAUM; VAN STEEN, 2007). Pretendia-se que a RPC se tornasse o mais parecida possível “[...] com as chamadas de procedimentos locais, sem nenhuma distinção na sintaxe entre uma chamada de procedimento local e um remoto” (COULOURIS *et al.*, 2013, p. 199).



O conceito de RPC foi apresentado pela primeira vez por Birrel e Nelson, em um artigo, em 1984, “[...] e abriu o caminho para muitos dos desenvolvimentos na programação de sistemas distribuídos utilizados atualmente”. Tais desenvolvimentos representaram um importante avanço intelectual (COULOURIS et al., 2013, p. 195).

Assim, o objetivo por trás da RPC está em garantir transparência de maneira simples. Isso é possível a partir da abstração na chamada de procedimentos em ambientes distribuídos, ou seja, o processo que efetuou uma chamada deve desconhecer o fato de que o processo chamado está executando em um dispositivo diferente (COULOURIS et al., 2013; TANENBAUM; VAN STEEN, 2007). Além disso, busca-se com a RPC eficiência e segurança na comunicação em ambientes distribuídos (DEITEL; DEITEL; CHOFFNES, 2005).

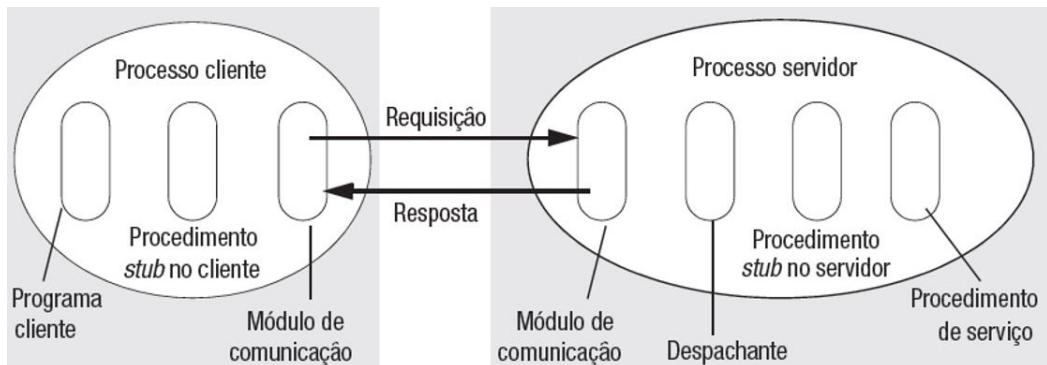
Isso entendido, você poderia agora se perguntar, como a RPC funciona na prática? De maneira resumida podemos dizer que a RPC permite que programas chamem procedimentos localizados em outras máquinas (TANENBAUM; VAN STEEN, 2007). Isso é possível porque uma máquina cliente emite uma chamada de procedimento remoto enviando parâmetros pela rede até a máquina servidora onde se encontra o procedimento requisitado. Esta, por sua vez, executa o procedimento e devolve o resultado ao cliente (COULOURIS et al., 2013; DEITEL; DEITEL; CHOFFNES, 2005). Tudo isso acontece como se fossem procedimentos sendo executados localmente - então está aí a transparência de que tanto falamos!

Veja o exemplo a seguir:

Quando um processo na máquina A chama um procedimento na máquina B, o processo chamador em A é suspenso, e a execução do procedimento chamado ocorre em B. Informações podem ser transportadas do chamador para quem foi chamado nos parâmetros e podem voltar no resultado do procedimento. Absolutamente nada de troca de mensagens é visível para o programador (TANENBAUM; VAN STEEN, 2007, p. 75).

Então, na prática, os componentes de software necessários para a implementação de RPC introduzem o conceito de *stub*. De maneira geral, um *stub* pode ser definido como “[...] um pedaço de código que transforma requisições que vêm pela rede em chamadas de procedimentos locais” (TANENBAUM; VAN STEEN, 2007, p. 77), ou seja, “[...] prepara dados de saída para transmissão e traduz dados de entrada para que possam ser interpretados corretamente” (DEITEL; DEITEL; CHOFFNES, 2005, p. 511). Desta forma, na RPC, sempre encontraremos um procedimento *stub* no cliente e um procedimento *stub* no servidor, conforme você pode perceber na Figura 5.

FIGURA 5 – FUNCIONAMENTO RPC



FONTE: Coulouris et al. (2013, p. 201)



O termo *stub* também pode ser encontrado na literatura como "apêndice". Então, em obras como a de Tanenbaum e Van Steen (2007), você encontrará referência a "apêndice de cliente" e "apêndice de servidor". Contudo, o termo *stub* ainda é o mais encontrado e por esta razão foi o termo utilizado neste livro.

O funcionamento da RPC ilustrada na Figura 5 pode ser resumido da seguinte forma (COULOURIS et al., 2013; DEITEL; DEITEL; CHOHNES, 2005; TANENBAUM; VAN STEEN, 2007):

- Para realizar uma RPC, o processo cliente chama o procedimento *stub*. O *stub* do cliente então empacota os parâmetros em uma mensagem, e realiza uma requisição via módulo de comunicação, para que esta seja enviada ao servidor.
- Ao receber a mensagem do *stub* do cliente, o SO do servidor a envia para o *stub* do servidor (o equivalente, no lado do servidor, ao *stub* do cliente). A mensagem então é desempacotada pelo *stub* do servidor que executa o procedimento local apropriado, e os valores de retorno para a mensagem de resposta são empacotados para que sejam enviados de volta ao cliente.

Repare que, graças à RPC, do ponto de vista do cliente, todo esse processo se parece com uma chamada de procedimento local e, de maneira semelhante, do ponto de vista do servidor, é como se ele estivesse sendo chamado diretamente pelo cliente. Ou seja, nem um nem outro ficam a par das etapas intermediárias ou da existência de uma rede.

Contudo, apesar de se mostrar uma solução eficiente no contexto da comunicação em sistemas distribuídos, há algumas complicações relacionadas à RPC que merecem ser mencionadas (COULOURIS *et al.*, 2013; DEITEL; DEITEL; CHOFFNES, 2005):

- A transparência da RPC pode ser limitada pelo fato de sua emissão e de seu correspondente *stub* do cliente se encontrarem em espaços de endereçamento de memória distintos, o que dificulta a passagem de ponteiros como parâmetros.
- Questões de desempenho e segurança apontam que a RPC ainda não se mostra uma solução suficiente na comunicação distribuída.
- RPC são mais vulneráveis às falhas do que chamadas locais, já que envolvem uma rede, outro computador e outro processo, aumentando as chances de que nenhum resultado seja recebido. A situação ainda se agrava quando, em caso de falha, torna-se impossível identificar se a mesma ocorreu na rede ou no processo servidor remoto.
- A latência de um procedimento remoto é muito maior do que a de um procedimento local.

3.2 MIDDLEWARE ORIENTADO A MENSAGEM

Conforme você estudou, RPC contribui fortemente para a comunicação dos sistemas distribuídos à medida que promovem sua transparência. Compreendemos que esta é verificada quando se garante a troca de mensagens entre uma máquina cliente e uma máquina servidora através de uma rede, e nenhuma e nem outra sabem disso.

Contudo, o fato de a comunicação se tornar transparente também evidencia outro aspecto: não conseguimos saber se o sistema para o qual faremos uma RPC (lado receptor) está executando no momento em que emitimos uma requisição. Por essa razão, tornam-se necessários serviços de comunicação alternativos (TANENBAUM; VAN STEEN, 2007).

Esses serviços alternativos à RPC são comumente chamados sistemas de comunicação orientados a mensagens, em geral construídos sobre a camada de transporte (TANENBAUM; VAN STEEN, 2007). Tais serviços são identificados como: soquetes, interface de passagem de mensagem (*Message Passing Interface* – MPI) e sistema de enfileiramento de mensagens ou *middleware* orientado a mensagens (MOM).

O primeiro dos serviços que apresentaremos a você é o soquete. Um soquete pode ser definido como uma camada de abstração utilizada pelo SO com um protocolo de comunicação específico com a finalidade de abrir um canal de comunicação entre dois dispositivos na rede (TANENBAUM; VAN STEEN, 2007). Em outras palavras, podemos dizer que um soquete é um terminal de comunicação, a partir do qual uma aplicação pode escrever dados que serão enviados e ler os dados que serão recebidos.

Você deve recordar, conforme estudado na Unidade 1, sobre a evolução dos sistemas computacionais e o surgimento de processadores de múltiplos núcleos. Essa evolução não somente permitiu o processamento paralelo, mas também fez com que fabricantes desenvolvessem seus próprios sistemas de comunicação, com hardwares proprietários.

Isso se converteu em um grande problema de incompatibilidade de mensagens transmitidas entre sistemas de fabricantes diferentes. Tornava-se então necessário um padrão para a troca de mensagens que fosse independente de hardware: surgia então a MPI.

A MPI foi projetada para atender aplicações paralelas. Isso é possível porque ela considera que a comunicação ocorre em grupos de processos, os quais poderão estar em execução ao mesmo tempo (TANENBAUM; VAN STEEN, 2007).

Finalmente, os MOM “[...] oferecem capacidade de armazenamento de médio prazo para mensagens, sem exigir que o remetente ou o receptor estejam ativos durante a transmissão” (TANENBAUM; VAN STEEN, 2007, p. 87).

Isso é possível porque, como você já deve imaginar, as mensagens são inseridas em fila, até que sejam entregues ao seu destino. Isso agrupa ao MOM uma característica que permite que a transferência de mensagens possa durar minutos, pois há independência de execução entre remetente e destinatário (TANENBAUM; VAN STEEN, 2007). Ou seja, não é necessário que o destinatário esteja em execução quando a mensagem é enviada para a sua fila, ou que o remetente esteja em execução quando o destinatário pegar a mensagem.

3.3 FLUXO DE DADOS

Os sistemas de comunicação que você estudou até este ponto neste livro trataram do envio de mensagens (pacotes de dados) em ambientes de sistemas distribuídos. De maneira mais simplista, falamos em como um sistema de e-mail, por exemplo, se comportaria.

Mas sabemos que em uma rede subjacente não existe somente a troca de mensagens, como os dados em um e-mail. Eventualmente pode haver a necessidade de se trocar uma quantidade muito maior de dados, como transmitir áudio e vídeo, como em uma videoaula, por exemplo. Esta é uma transmissão dependente de tempo (TANENBAUM; VAN STEEN, 2007).

Esse tipo de transmissão é definido como fluxo de dados, ou *stream*. Essa forma de comunicação em sistemas distribuídos se distingue das demais por exigir larguras de bandas muito altas, latência e confiabilidade (COULOURIS *et al.*, 2013).

Desta forma, a qualidade de serviço para a transmissão de dados deve ser garantida, o que, em geral, não se consegue pela internet. Você já deve ter passado pela experiência de estar assistindo a uma videoaula pelo Youtube e esta ser interrompida várias vezes para ser carregada. Em geral, a qualidade é obtida,

[...] com o estabelecimento de um canal de comunicação de um fluxo multimídia da origem para o destino, com uma rota predefinida pela rede, com um conjunto de recursos reservados em cada nó pelo qual ele passará [...]. Os dados podem, então, passar pelo canal, do remetente para o destinatário, na velocidade exigida (COULOURIS *et al.*, 2013, p. 90).

3.4 MULTICASTING

Agora que você já estudou as diversas técnicas de comunicação em sistemas distribuídos, pôde compreender o esforço necessário para a sua implementação e a necessidade relacionada à garantia de qualidade, transparência e disponibilidade da informação.

Nesse sentido, um tópico de interesse, nesse contexto da comunicação em sistemas distribuídos, está na possibilidade de efetuar a troca de mensagens não com somente um, mas com vários destinatários simultaneamente. A este tipo de operação dá-se o nome de comunicação *multicast* ou *multicasting* (COULOURIS *et al.*, 2013; TANENBAUM; VAN STEEN, 2007).

De maneira geral, no *multicasting* é possível que “[...] os nós se organizem em uma rede de sobreposição que então é usada para disseminar informações para seus membros” (TANENBAUM; VAN STEEN, 2007, p. 100).

Assim, no tocante a infraestrutura *multicast* para sistemas distribuídos, são levadas em conta algumas características (COULOURIS *et al.*, 2013):

- **Tolerância a falhas:** alcançada a partir da replicação dos serviços em grupos de servidores. Desta forma, mesmo se um dos membros da rede falhar, as requisições dos clientes ainda poderão ser atendidas. Este é um tópico de grande interesse em seu estudo, por isso será retomado mais adiante no Tópico 2.
- **Melhor desempenho:** obtido através da replicação de dados (que também será retomada em detalhes no Tópico 2).
- **Propagação de notificação de evento:** trata-se da utilização de *multicast* para notificar um grupo. É o que acontece, por exemplo, quando você faz uma atualização de seu status no Facebook, e todos os seus amigos recebem a notificação desta alteração.

RESUMO DO TÓPICO 1

Neste tópico, você aprendeu que:

- A comunicação entre processos de sistemas distribuídos ocorre sobre as diferentes estruturas de redes, cujas características afetam seu projeto.
- Os diferentes tipos de redes de computadores, bem como seus protocolos, estabelecem quais são as tecnologias empregadas na transmissão de dados.
- A transmissão de informações através de uma rede é chamada de transmissão de mensagens, e ela se dá entre um emissor ou remetente (origem) e o receptor ou destinatário (destino).
- Os protocolos são regras que estabelecem os padrões para trocas de mensagens entre os diferentes componentes em um sistema distribuído, especificando o formato dos dados da mensagem, e a sequência das mensagens.
- As redes devem apresentar características como: desempenho, escalabilidade, confiabilidade, segurança, mobilidade e *multicasting* para o funcionamento adequado dos sistemas distribuídos.
- Interoperabilidade entre aplicações é uma importante característica dos sistemas distribuídos, pois permite a interação de componentes de software residentes em diferentes plataformas de hardware.
- A comunicação entre processos pode ser síncrona e assíncrona. A diferença entre os tipos de comunicação está no comportamento das operações de *send* e *receive*.
- A RPC tem como objetivo manter a transparência de comunicação em sistemas distribuídos.
- RPC introduz o conceito de *stub*, uma espécie de interface que transforma uma requisição que vem pela rede em uma chamada local.
- Sistemas orientados a mensagem são serviços alternativos à RPC, podendo ser: soquetes, MPI e MOM.
- Soquetes são uma camada de abstração que realiza o papel de terminal de comunicação para as aplicações.

- MPI atende aplicações paralelas, ou seja, grupos de processos que poderão estar simultaneamente em execução.
- MOM implementa o enfileiramento de mensagens, desta forma, remetente ou receptor não precisam estar ativos durante a transmissão.
- Fluxos de dados ou *stream* representam um tipo muito particular de comunicação, associado à transmissão de áudio e vídeo, exigindo altas larguras de banda, latência e confiabilidade dos sistemas distribuídos.
- *Multicast* envolve a comunicação entre vários destinatários simultaneamente.
- Para comunicação em *multicast* devem-se considerar características como: tolerância a falhas, melhor desempenho e propagação de notificações.



1 Está claro que os sistemas distribuídos precisam das redes de computadores, ou até mesmo redes de dispositivos, para o seu funcionamento. Além de elas oferecerem uma infraestrutura de conectividade, também oferecem uma infraestrutura de comunicação para as entidades envolvidas nas trocas de mensagens no ambiente distribuído. Com isso em mente, avalie as sentenças relacionadas aos protocolos de comunicação classificando com V as sentenças verdadeiras e F as falsas:

- () Nas trocas de mensagens em um ambiente distribuído, as entidades (nodos) envolvidas nessa troca, ora podem se comportar como emissores, e ora podem se comportar como receptores.
- () O caminho que as mensagens percorrem nas camadas de protocolos é diferente nos sistemas distribuídos quando comparadas aos sistemas tradicionais cliente/servidor devido a sua natureza mais complexa.
- () O caminho que as mensagens percorrem nas camadas de protocolos é a mesma nos sistemas distribuídos quando comparadas aos sistemas tradicionais cliente/servidor independentemente de sua natureza mais complexa.
- () Nas trocas de mensagens em um ambiente distribuído, as entidades (nodos) envolvidas nessa troca, quem atua como emissor sempre será um emissor, e da mesma forma quem se comporta como receptor sempre será um receptor.

Agora, assinale a alternativa CORRETA:

- a) () V – F – F – V.
- b) () F – V – V – F.
- c) () F – V – F – V.
- d) () V – F – V – F.

2 Nos ambientes de sistemas distribuídos não há como processos serem executados, ou recursos serem acessados sem o envio/recebimento de pacotes, e, para isso, as entidades lançam mão de duas operações básicas envio e recebimento. Com isso em mente, avalie as sentenças relacionadas ao conceito de sincronização classificando com V as sentenças verdadeiras e F as falsas:

- () Na comunicação síncrona uma ação de impedimento, nas operações básicas (*send/receive*), impede as ações da entidade origem da mensagem até que sua requisição seja aceita.
- () As comunicações síncrona e assíncrona possuem características semelhantes independentes da natureza de comunicação exigida por um recurso distribuído (serviço, aplicação etc.) quando da solicitação desse recurso por uma entidade emissora e a resposta dada por uma entidade receptora. Dessa forma, toda comunicação síncrona pode ser assíncrona e vice-versa.

- () Na comunicação assíncrona uma ação de impedimento nas operações básicas (*send/receive*), impede as ações da entidade origem da mensagem até que sua requisição seja aceita.
- () As comunicações síncrona e assíncrona dependem da natureza de comunicação exigida por um recurso distribuído (serviço, aplicação etc.) quando da solicitação desse recurso por uma entidade emissora e a resposta dada por uma entidade receptora.

Agora, assinale a alternativa CORRETA:

- a) () V – V – V – F.
b) () V – F – F – V.
c) () F – V – V – V.
d) () V – F – V – F.

3 Está claro que os sistemas distribuídos precisam das redes de computadores, ou até mesmo redes de dispositivos, para o seu funcionamento. Além de elas oferecerem uma infraestrutura de conectividade, também oferecem uma infraestrutura de comunicação para as entidades envolvidas nas trocas de mensagens no ambiente distribuído. Com isso em mente, avalie as sentenças relacionadas aos protocolos de comunicação:

- I- Os protocolos de comunicação são responsáveis por garantir que uma mensagem enviada por uma entidade emissora seja recebida pela entidade receptora.
- II- Nas camadas do modelo OSI, em número de 7, as camadas mais baixas são mais próximas das aplicações dos usuários e as camadas mais altas mais próximas dos meios físicos da rede.
- III- Os protocolos de comunicação são organizados logicamente em camadas sobrepostas uma a outra, na qual cada uma delas possui um conjunto de regras específicas para a recepção, tratamento e repasse de uma mensagem para outra camada.
- IV- Nas camadas do modelo OSI, em número de 5, as camadas mais baixas representam camadas mais próximas dos meios físicos das redes e as mais altas mais próximas das aplicações dos usuários.

Agora, indique a alternativa CORRETA:

- a) () As sentenças I e III estão corretas.
b) () As sentenças II e IV estão corretas.
c) () As sentenças I e II estão corretas.
d) () As sentenças III e IV estão corretas.

4 Nos ambientes de sistemas distribuídos não há como processos serem executados, ou recursos serem acessados sem o envio/recebimento de pacotes, e, para isso, as entidades lançam mão de duas operações básicas, envio e recebimento. Com isso em mente, avalie as sentenças relacionadas ao conceito de sincronização:

- I- Na comunicação síncrona as entidades emissora e receptora trocam mensagens sem a necessidade de um evento de impedimento, logo, uma operação de envio não impede a entidade emissora de continuar enviando transmissões à entidade receptora.
- II- Na comunicação assíncrona as entidades emissora e receptora trocam mensagens sem a necessidade de um evento de impedimento, devido à utilização de um repositório intermediário (*buffer*) no qual as mensagens são armazenadas.
- III- Na comunicação assíncrona as entidades emissora e receptora trocam mensagens sem a necessidade de um evento de impedimento, logo, uma operação de envio não impede a entidade emissora de continuar enviando transmissões à entidade receptora.
- IV- Na comunicação síncrona, ou mesmo na assíncrona, as mensagens das entidades emissoras são produzidas sem a interferência de eventos de impedimento, logo, as operações de envio e de recebimento não são influenciadas por esse tipo de evento.

Agora, indique a alternativa CORRETA:

- a) () As sentenças I e III estão corretas.
- b) () As sentenças II e IV estão corretas.
- c) () As sentenças II e III estão corretas.
- d) () As sentenças III e IV estão corretas.

5 Nos ambientes de sistemas distribuídos não há como processos serem executados, ou recursos serem acessados sem o envio/recebimento de pacotes, e, para isso, as entidades lançam mão de duas operações básicas, envio e recebimento, sendo que esse envio/recebimento deve ser transparente, isto é, como se ele estivesse acontecendo localmente. Dessa forma, avalie a sentença de afirmação e a sentença de explicação relacionadas ao modelo de comunicação RPC envolvido na construção de sistemas distribuídos.

I- O modelo RPC tem como objetivo tornar o mais transparente possível as chamadas remotas como se fossem chamadas a serviços, ou recursos, locais. Isso deve ocorrer sem que haja qualquer distinção na sintaxe de parâmetros entre uma chamada de procedimento local e um remoto.

PORQUE

II- Em um ambiente distribuído um defeito, erro ou falha, podem ocorrer e comprometer a eficiência, e eficácia, isto é, o próprio comportamento do sistema. Logo, o sistema deve ser o que se chama de tolerante a esses eventos como forma de manter a qualidade de execução de seus serviços.

Agora, assinale a alternativa que apresenta a resposta CORRETA:

- a) () As sentenças I e II representam proposições verdadeiras, mas a II não é uma justificativa correta da I.
- b) () As sentenças I e II não representam proposições verdadeiras.
- c) () A sentença I é uma proposição verdadeira, porém a II é uma proposição falsa.
- d) () As sentenças I e II representam proposições verdadeiras, e a II é uma justificativa correta da I.

CONSISTÊNCIA E TOLERÂNCIA A FALHAS

1 INTRODUÇÃO

Em sua trajetória sobre sistemas distribuídos, você já estudou que estes se tratam de um conjunto de sistemas autônomos, que estão interconectados por uma estrutura de rede. Conforme você acabou de estudar no Tópico 1 desta Unidade 3, a comunicação entre os diferentes dispositivos espalhados neste ambiente de rede deve se dar de maneira transparente, como se estivéssemos trabalhando em um sistema local.

Além da transparência, características como escalabilidade, segurança e tolerância a falhas (conceituadas no Tópico 1 da Unidade 2), também são essenciais aos sistemas distribuídos.

Neste tópico, você será convidado a aprofundar um pouco mais seus conhecimentos sobre as características dos sistemas distribuídos e apresentaremos a você um tópico de grande interesse: a replicação de dados. Veremos que a replicação é elemento crítico para promover a disponibilidade e a tolerância a falhas em sistemas distribuídos; além de melhorar confiabilidade e desempenho (COULOURIS *et al.*, 2013; TANENBAUM; VAN STEEN, 2007).

Consistência também representará um papel importante em nossos estudos neste Tópico 2, já que uma vez que se fazem réplicas de dados, estas devem se manter consistentes.

2 REPLICAÇÃO E CONSISTÊNCIA

Iniciaremos, então, nossos estudos sobre replicação, compreendendo o que é e por que é utilizada. De maneira geral, a replicação pode ser definida como a ação de manter cópias de dados em vários computadores. Ao se manter estas cópias, aprimora-se a eficácia dos sistemas distribuídos, já que se obtêm ganhos em desempenho, disponibilidade, confiabilidade e também, tolerância a falhas (COULOURIS *et al.*, 2013; TANENBAUM; VAN STEEN, 2007).

Em um contexto prático, a replicação pode ser exemplificada da seguinte forma: “[...] o armazenamento de recursos de servidores Web na cache dos navegadores [...] é uma forma de replicação, pois os dados mantidos nas caches [...] são réplicas uns dos outros” (COULOURIS *et al.*, 2013, p. 766). Desta forma, ganha-se em desempenho, por exemplo.

Agora que você já compreendeu o conceito de replicação, é importante também reconhecer as razões básicas para se replicar dados, ou seja, o porquê de sua utilização. A literatura estabelece que as razões para se replicar envolvem: melhorar o desempenho, aumentar a confiabilidade, aumentar sua disponibilidade ou para que seja tolerante a falhas (COULOURIS *et al.*, 2013; TANENBAUM; VAN STEEN, 2007).

Iniciaremos falando sobre a confiabilidade. Imagine que você esteja trabalhando em seu notebook, que é seu ambiente de armazenamento primário. Se ele falhar, seu trabalho estará comprometido. Por outro lado, se seus dados foram replicados, será possível continuar trabalhando a partir de uma réplica. Além de permitir a continuidade do trabalho, a replicação também oferece proteção contra dados corrompidos.

Além da confiabilidade, usuários em geral exigem alta disponibilidade nos serviços. Isso significa que usuários são intolerantes a atrasos, e esperam não só que os serviços estejam disponíveis, mas que também tenham tempo de resposta “razoável”. Há fatores que afetam a disponibilidade de serviços em sistemas distribuídos, tais como falhas de servidor e particionamento de rede. Contudo, a replicação mostra-se uma ótima solução para ambos os casos.

Com relação a falhas de servidor, a replicação pode ser considerada a primeira técnica para se manter a disponibilidade dos dados automaticamente, e não é difícil entendermos o porquê. Perceba que se os dados estão replicados em vários servidores, no caso da falha de algum deles, o software cliente pode acessar os dados em um servidor alternativo.



Há uma diferença entre a replicação dos servidores e o uso de cache (que mencionamos no exemplo logo no início desta seção). As caches [...] não contêm necessariamente conjuntos de objetos [...] em sua totalidade”, o que faz com que seu uso não melhore a disponibilidade (COULOURIS *et al.*, 2013, p. 767).

Já o problema de particionamento da rede ocorre quando, por exemplo, “[...] usuários móveis desconectam deliberadamente seus computadores ou, ao se moverem, podem ser desconectados involuntariamente de uma rede sem fio” (COULOURIS *et al.*, 2013, p. 767).

A replicação de dados entre dispositivos também pode ser aplicada para que se tenha um sistema tolerante a falhas, cuja principal característica é garantir um comportamento correto, ou gerenciar seus componentes de modo a corrigir eventuais falhas. Sobre os conceitos relacionados à tolerância a falhas em sistemas distribuídos, você os estudará mais adiante.

Finalmente, outra razão para a replicação de dados é a melhoria de desempenho. Uma forma bastante comum para se ganhar desempenho (e que já mencionamos anteriormente em um exemplo) é o uso de caches de clientes e servidores. Conforme já ilustramos, cópias de recursos web em caches de navegadores evitam a latência de novas buscas a esses recursos.

A replicação torna-se importante também para o desempenho de sistemas distribuídos quando estes precisam ser escalonados (aumentados) em número ou área (geograficamente falando) (TANENBAUM; VAN STEEN, 2007).

Escalona-se um sistema em número quando o número de servidores é replicado. Isso ocorre quando uma grande quantidade de processos é gerenciada por um único servidor. Replicando-se as máquinas servidoras, o trabalho de processamento é dividido entre elas, melhorando-se, consequentemente, o desempenho.

Um sistema também pode ser escalonado em razão de sua área. Nesta situação, máquinas clientes de um determinado processo podem estar geograficamente distantes. Neste caso a replicação se daria ao colocar cópias de tais processos próximas de quem as está utilizando, diminuindo o tempo de acesso aos dados e, consequentemente, aumentando-se o desempenho.



Requisito importante para a replicação é sua transparência. Os clientes não devem ter consciência da existência de várias cópias dos dados (COULOURIS *et al.*, 2013).

Agora que você já compreendeu porque é importante utilizar a replicação de dados em sistemas distribuídos, passaremos a outro aspecto relacionado a ela e que, se não levado em conta, pode se converter em um problema. Trata-se da consistência.

Entendemos que conjuntos de dados são consistentes quando todas as suas cópias (réplicas) possuem os mesmos valores. Desta forma, sempre que uma cópia é modificada, ela se torna diferente das demais, sendo necessário modificar todas as outras cópias para garantir sua consistência.

Retomaremos mais uma vez o exemplo da *cache* do navegador *web*. Se um usuário requisitar novamente uma página que acessou anteriormente, o navegador automaticamente retorna a réplica local. Porém, se entre um acesso e outro a página sofreu alguma modificação, a cópia na *cache* não terá sido atualizada (TANENBAUM; VAN STEEN, 2007). Então, a replicação em *cache* melhorou o desempenho, mas gerou-se um problema de consistência.

Perceba então que, sempre que uma cópia é modificada em algum lugar, todas as demais réplicas também precisarão ser, caso contrário se tornarão inconsistentes. “O que determina o preço da replicação é exatamente quando e como essas modificações precisam ser executadas” (TANENBAUM; VAN STEEN, 2007, p. 166).

Em outras palavras, é preciso que haja sincronização entre as réplicas. Contudo, executar um processo de sincronização (global) fatalmente afetará o desempenho de nosso sistema distribuído. Para resolver este problema envolvendo desempenho versus consistência, verificou-se que a melhor solução pode ser “relaxar” as regras de consistência, ou seja, minimizar os requisitos de atualizações (TANENBAUM; VAN STEEN, 2007).

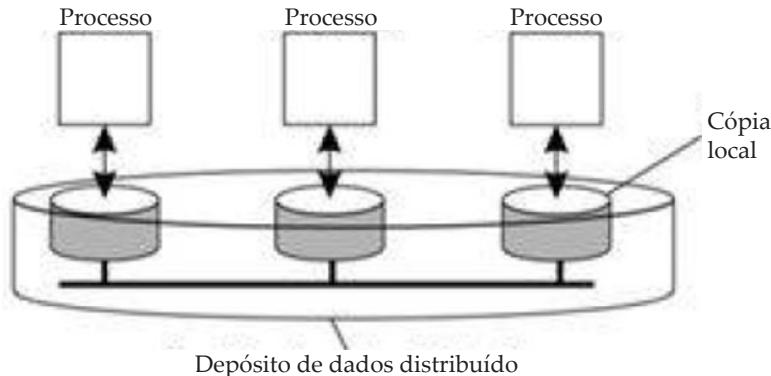
Ao se fazer isso, pode ocorrer que nem todas as cópias sejam iguais em todos os locais, tudo dependerá da necessidade ou da finalidade dos dados. Assim, para que possamos atestar até qual ponto relaxar as regras de consistência, vale uma verificação dos modelos de consistência disponíveis.

2.1 MODELOS DE CONSISTÊNCIA

Conforme você estudou até aqui, o conceito de consistência nos diz que, dado um conjunto de cópias de dados, todas devem ser iguais. Essa definição nos abre caminho para agregarmos mais alguns conceitos ao seu estudo. O primeiro diz respeito às cópias ou réplica dos dados. Assim, quando nos referimos à existência de réplicas, estamos nos referindo à realização de operações de leitura e escrita de dados compartilhados, o que nos leva ao segundo conceito. Os dados compartilhados estão disponíveis para leitura e escrita a partir do que chamamos de **depósito de dados** (TANENBAUM; VAN STEEN, 2007).

Depósito de dados é um termo genérico usado para denominar memória, bancos de dados e/ou sistemas de arquivos compartilhados (distribuídos), ou seja, um depósito de dados “[...] pode estar distribuído fisicamente por várias máquinas” (TANENBAUM; VAN STEEN, 2007, p. 167), conforme você pode verificar na Figura 6.

FIGURA 6 – REPLICAÇÃO EM UM DEPÓSITO DE DADOS DISTRIBUÍDO



FONTE: Tanenbaum e Van Steen (2007, p. 167)

Agora que já esclarecemos estes dois conceitos, podemos definir um **modelo de consistência**. Este se trata, em termos gerais, de um conjunto de regras estabelecido entre processos e o depósito de dados. Em outras palavras, se os processos obedecerem às regras estabelecidas no modelo de consistência, o depósito de dados retornará exatamente o que se espera (TANENBAUM; VAN STEEN, 2007).

Nesse sentido, há diferentes modelos de consistência, que variam de acordo com os tipos de restrições estabelecidas. Esses modelos podem ser organizados em dois grandes grupos: os modelos de consistência baseados em dados, e os modelos de consistência baseados no cliente.

Os modelos de consistência baseados em dados “[...] visam fornecer uma visão consistente de um depósito de dados no âmbito de um sistema” (TANENBAUM; VAN STEEN, 2007, p. 174) e envolvem a consistência contínua, a consistência sequencial e a consistência causal (TANENBAUM; VAN STEEN, 2007):

- **Consistência contínua:** este modelo estabelece três diferentes formas de tolerar inconsistências. A primeira diz respeito ao desvio em valores numéricos entre réplicas; a segunda trata do desvio em idade entre réplicas; e, finalmente, a terceira considera o desvio em relação à ordenação de operações de atualização. A **consistência em relação a desvio de valores numéricos** leva em conta a “semântica numérica de dados”, ou seja, se as réplicas armazenam um valor numérico monetário, por exemplo, os dados seriam considerados

consistentes entre réplicas ainda que apresentassem uma diferença de valor (desde que não violasse os limites especificados). Já a **consistência em relação ao desvio de idade entre réplicas** considera a última vez que uma réplica foi atualizada. Este tipo de consistência tolera o fornecimento de dados “antigos”, que são atualizados de tempos em tempos. Por fim, a **consistência em relação à ordenação de operações de atualização** tolera a diferente ordenação das atualizações nas diferentes réplicas. Isso ocorre da seguinte forma: as atualizações são aplicadas provisoriamente a uma cópia local que fica à espera de um acordo para a atualização das demais réplicas.

Os próximos dois modelos de consistência baseada em dados surgiram da verificação de que, em sistemas distribuídos (e computação paralela), vários processos compartilharão recursos simultaneamente, isso também ocorrerá no caso da replicação desses recursos compartilhados.

- **Consistência sequencial:** importante modelo de consistência baseado em dados, estabelece o seguinte: “[...] quando processos executam concorrentemente em máquinas [...] diferentes, qualquer intercalação válida de operações de leitura e de escrita é um comportamento aceitável, mas todos os processos veem a mesma intercalação de operações” (TANENBAUM; VAN STEEN, 2007, p. 170). Em outras palavras, a consistência sequencial garante que o resultado de operações de dados (leitura e escrita) seria o mesmo, se fossem executadas por todos os processos ou por um processo individualmente, na mesma ordem sequencial.
- **Consistência causal:** esse modelo de consistência toma em conta a causa que leva à execução de um determinado evento. Assim, um depósito de dados é consistente por causalidade quando se conhece a dependência entre operações, ou seja, quando se sabe qual operação é dependente de quais outras. Além disso, nesse tipo de consistência, operações de escrita devem ser lidas na mesma ordem por todos os processos.

A partir de outra perspectiva, os modelos de consistência centrados no cliente concedem a um único cliente consistência de acesso a um depósito de dados. Os modelos de consistência centrados no cliente envolvem: leitura monotônica, escrita monotônica, leitura das suas escritas e escrita após a leitura (TANENBAUM; VAN STEEN, 2007):

- **Leitura monotônica:** esse modelo de consistência garante que, se um processo leu um valor no depósito de dados, uma operação de leitura executada por esse processo retornará o mesmo valor ou um valor mais recente.
- **Escrita monotônica:** esse modelo garante que as operações de escrita sejam replicadas na ordem correta para todas as cópias do depósito de dados. Ou seja, uma nova operação de escrita sobre uma cópia somente será realizada após o término da operação de escrita anterior.
- **Leitura das suas escritas:** esse modelo de consistência está diretamente relacionado ao modelo de leitura monotônica. A consistência por leitura das suas escritas estabelece que um processo que realizou uma operação de escrita, sempre realizará uma operação de leitura na sequência.

- **Escrita após leitura:** neste modelo “[...] as atualizações são propagadas como resultado de operações de leitura precedentes” (TANENBAUM; VAN STEEN, 2007, p. 178). Em outras palavras, um processo realizará uma operação de escrita sobre a cópia atualizada do último valor lido por esse processo.

3 TOLERÂNCIA A FALHAS

Já compreendemos a importância da replicação de dados para os sistemas distribuídos e como problemas de consistência, ou seja, a garantia de que os mesmos dados serão encontrados nas várias réplicas, podem ser resolvidos.

Chegamos agora a outro aspecto de grande importância no seu estudo: o conceito de falha parcial, existente para sistemas distribuídos, e como se recuperar dessas falhas. A partir de agora você entenderá como tornar sistemas distribuídos tolerantes a falhas.

Iniciaremos nosso estudo conhecendo três conceitos importantes: as diferenças entre defeito, erro e falha (TANENBAUM; VAN STEEN, 2007):

- **Defeito:** diz-se que um sistema está com defeito quando não cumpre aquilo que deveria fazer. Isso acontece o tempo todo em nosso cotidiano. Por exemplo, se você chega em casa em um dia quente e liga seu aparelho de ar-condicionado, espera que ele comece a resfriar o ambiente. Se depois de um tempo ligado você não perceber qualquer alteração na temperatura do ambiente, logo conclui: “Está com defeito!”. De maneira semelhante, um sistema distribuído apresenta defeito quando não fornece os serviços aos seus usuários.
- **Erro:** representa um estado de um sistema. Quando você envia um documento do Word para impressão e esta não ocorre, normalmente o sistema apresenta na tela a informação de que houve um erro na impressão. Esse erro não necessariamente está associado a um defeito em sua impressora. Porém, erros normalmente tem uma causa, à qual denominamos falha.
- **Falha:** é a causa de um erro, e, em quaisquer circunstâncias, especialmente em nosso estudo sobre sistemas distribuídos, identificar “o que” causou um erro é muito importante. Em nosso exemplo anterior, o erro de impressão pode ter sido causado porque sua impressora estava desligada, ou porque não havia folhas de papel para prosseguir com o serviço. Essas são falhas simples de serem resolvidas, mas há falhas mais complexas que podem interferir na continuidade do trabalho.

Nesse sentido, podemos retomar o tema destacado no início desta seção, quando abordamos o termo “falha parcial”. Esse termo diferencia sistemas distribuídos de um sistema computacional local, pois estabelece que, caso haja uma falha em um dos componentes do sistema, sem afetar os demais componentes, sua operação adequada ainda é permitida (TANENBAUM; VAN STEEN, 2007).

Conforme você já estudou, isso é possível graças à replicação, ao desenvolvimento de mecanismos de consistência dos dados em diferentes dispositivos, que garantem a transparência ao usuário, e à possibilidade de se recuperar dessas falhas parciais (COULOURIS *et al.*, 2013; DEITEL; DEITEL; CHOHNES, 2005; TANENBAUM; VAN STEEN, 2007).

De maneira geral, falhas podem ser evitadas, removidas ou previstas. Contudo, nosso interesse está na possibilidade de nosso sistema ser “tolerante” a falhas, ou seja, possibilitar que um sistema distribuído possa detectar e reagir a falhas, e continue provendo seus serviços (DEITEL; DEITEL; CHOHNES, 2005; TANENBAUM; VAN STEEN, 2007).

3.1 SISTEMAS CONFIÁVEIS

A literatura nos diz que há forte relação entre a confiança de um sistema e sua capacidade de ser tolerante a falhas. Nesse sentido, um sistema confiável, desejavelmente, apresenta as seguintes características (TANENBAUM; VAN STEEN, 2007):

- **Disponibilidade:** diz-se que um sistema é disponível quando está pronto para uso imediato.
- **Confiabilidade:** esta é uma característica de sistemas que podem funcionar um longo período de tempo, sem falhas.



Há uma diferença importante entre disponibilidade e confiabilidade. Se um sistema ficar fora do ar por um curtíssimo período de tempo (por exemplo, um milissegundo) a cada hora, ele terá alta disponibilidade (provavelmente você nem perceba que ele ficou fora do ar), mas será de baixa confiabilidade (você confiaria em um sistema que cai uma vez por hora? São 24 vezes por dia!).

- **Segurança:** um sistema é considerado seguro se, mesmo após uma parada, nenhuma situação crítica venha a acontecer. Retomaremos o contexto de segurança no Tópico 3 deste seu material didático.
- **Capacidade de manutenção:** diz respeito à facilidade de se consertar uma falha em um sistema. A alta capacidade de manutenção de um sistema (se as falhas puderem ser detectadas e recuperadas automaticamente) lhe confere alta disponibilidade.

3.2 TIPOS DE FALHAS

“A construção de sistemas confiáveis está intimamente relacionada com o controle de falhas” (TANENBAUM; VAN STEEN, 2007, p. 195), por essa razão é importante que saibamos identificar as diferentes classificações e tipos de falhas. A classificação está relacionada à ocorrência da falha e sua possível frequência. Já os tipos de falhas determinam a falha em si.

De modo geral as falhas são classificadas como: transientes, intermitentes ou permanentes (TANENBAUM; VAN STEEN, 2007):

- **Falhas transientes:** são caracterizadas por ocorrerem uma vez e depois desaparecerem. Falhas transientes não acontecem novamente se a operação for repetida.
- **Falhas intermitentes:** esse tipo de falha acontece e desaparece como se tivesse “vontade própria”, tornando difícil seu diagnóstico.
- **Falha permanente:** “[...] continua a existir até que o componente faltoso seja substituído” (TANENBAUM; VAN STEEN, 2007, p. 195).

Já os tipos de falha podem ser vistos no Quadro 2.

QUADRO 2 – TIPOS DE FALHAS

Tipo de falha	Descrição
Falha por queda	O servidor para de funcionar, mas estava funcionando corretamente até parar.
Falha por omissão	O servidor não consegue responder a requisições que chegam.
• Omissão de recebimento • Omissão de envio	O servidor não consegue receber mensagens que chegam. O servidor não consegue enviar mensagens.
Falha de temporização	A resposta do servidor se encontra fora do intervalo de tempo.
Falha de resposta	A resposta do servidor está incorreta
• Falha de valor • Falha de transição de estado	O valor da resposta está errado O servidor se desvia do fluxo de controle correto
Falha arbitrária	Um servidor pode produzir respostas arbitrárias em momentos arbitrários

FONTE: Adaptado de Tanenbaum e Van Steen (2007)

3.3 DETECÇÃO E MASCARAMENTO DE FALHAS

Até este ponto você já estudou e compreendeu o que é uma falha e seus tipos, e a importância de se ter sistemas tolerantes a falhas.

Contudo, se um erro em um sistema ocorre, é imprescindível que saibamos identificar sua causa, ou seja, saibamos detectar sua falha. A detecção de uma falha contribui para seu mascaramento.

Então você deve estar se perguntando: “Por que mascarar uma falha?”. Lembre-se que nosso estudo aqui está empenhado em tornar um sistema tolerante a falhas, ou seja, fazer com que ele continue funcionando apesar da existência de falhas parciais. Nesse sentido, devemos então tentar ocultar (mascarar) as falhas ocorridas em um processo, dos demais.

Para mascarar falhas é preciso, antes de qualquer coisa, detectá-las. É pela detecção que iniciaremos os estudos nessa seção.

Considerada fundamental para o projeto de sistemas distribuídos, a detecção de falhas estabelece que, dado um grupo de processos, é preciso ser capaz de detectar quando um membro do grupo falhou (TANENBAUM; VAN STEEN, 2007).

Pode-se dizer que existem, basicamente, dois mecanismos para um subsistema de detecção de falhas (TANENBAUM; VAN STEEN, 2007):

- 1- Processos enviam mensagens uns aos outros, esperando respostas.
- 2- Os processos esperam, passivamente, pela entrada de mensagens.

Assim, pode-se dizer que a detecção de falhas se resume à “[...] utilização de um mecanismo de esgotamento de temporização para verificar se um processo falhou” (TANENBAUM; VAN STEEN, 2007, p. 202). Contudo, na prática, uma abordagem de esgotamento de temporização pode gerar problemas, por isso soluções alternativas como a disseminação de informações por *gossip* tornam-se mais atraentes.



Gossiping é uma técnica a partir da qual processos informam sua disponibilidade de serviço periodicamente. Essa informação é então disseminada pela rede.

Uma vez detectada a falha, é importante então que consigamos mascará-la. Para tanto, podemos contar com técnicas de redundância. Há três tipos de redundância: de informação, de tempo e física (TANENBAUM; VAN STEEN, 2007).

- **Redundância de informação:** para a recuperação de eventuais bits deteriorados na mensagem, bits extras são adicionados. Um exemplo de redundância de informação é o uso do código de *Hamming*.
- **Redundância de tempo:** reexecução de uma ação caso necessário. Esse tipo de redundância é muito útil em caso de falhas transientes ou intermitentes.
- **Redundância física:** em caso de perda ou mau funcionamento de algum componente do sistema, equipamentos e/ou processos extras são adicionados. Esse tipo de redundância pode ocorrer tanto em nível de hardware quanto de software.



Código de *Hamming* é uma técnica que "[...] usa bits de paridade para verificar erros na transmissão de dados de discos e corrigi-los, se possível" (DEITEL; DEITEL; CHOHNES, 2005, p. 363).

3.4 RECUPERAÇÃO DE ERROS

Conforme você estudou anteriormente, a falha em um sistema é a causa de erros. A recuperação de erro, ou seja, a recuperação do processo sobre o qual a falha ocorreu para seu estado correto, é essencial para um sistema tolerante a falhas.

Desta forma, a recuperação de um erro pode ser definida como a substituição de um estado errôneo de um processo, por um estado correto ou livre de erro (TANENBAUM; VAN STEEN, 2007).

Há, basicamente, duas formas de recuperação de erro: retroativa e para frente (TANENBAUM; VAN STEEN, 2007):

- **Recuperação retroativa:** nesse caso, o sistema que se encontra em estado de erro é levado para um estado que estava correto anteriormente. Para que se conheça este estado anteriormente correto, de tempos em tempos o estado do sistema é registrado, gerando o que chamamos de ponto de verificação. Esta equivale a uma situação típica de restauração de backups.
- **Recuperação para frente:** é o oposto da recuperação retroativa. Aqui, ao invés de o sistema ser retornado ao último ponto de referência, tenta-se levá-lo para um novo estado correto.

RESUMO DO TÓPICO 2

Neste tópico, você aprendeu que:

- Replicação é a ação de manter cópias de dados em vários dispositivos.
- A partir da replicação obtém-se benefícios para os sistemas distribuídos como: desempenho, disponibilidade, confiabilidade e tolerância a falhas.
- A replicação garante a confiabilidade em casos de falhas de servidor e particionamento de rede, além de melhorar o desempenho quando sistemas distribuídos precisam ser escalonados.
- Consistência prevê que um conjunto de dados e suas réplicas possuem os mesmos valores.
- Manter a consistência entre réplicas é um processo complexo e pode afetar o desempenho do sistema distribuído se não houver uma avaliação adequada das regras de consistência.
- Um modelo de consistência é um conjunto de regras estabelecido entre processos e depósito de dados. Modelos de consistência podem ser baseados em dados ou centrados no cliente.
- **Defeito** é quando o sistema não cumpre o prometido; **erro** representa um estado do sistema; e **falha** é a causa do erro.
- Falha parcial caracteriza sistemas distribuídos no sentido de existir falhas em um componente do sistema, sem afetar os demais.
- Um sistema tolerante a falhas implica na capacidade do sistema de detectar e reagir a falhas e continuar provendo seus serviços.
- Falhas são classificadas como transientes, intermitentes ou permanentes.
- Os tipos de falhas podem ser: por queda, por omissão, de temporização, de resposta e arbitrárias.
- Detecção de falhas é fundamental no projeto de construção de sistemas distribuídos.

- Os mecanismos de detecção de falhas envolvem uma forma ativa (processos enviam mensagens uns aos outros esperando resposta), e uma forma passiva (os processos esperam pela entrada de mensagens).
- Mascarar falhas permite que o sistema continue funcionando apesar da existência destas, pois as oculta dos demais processos.
- Redundância é uma técnica de mascaramento, podendo ser de três tipos: informação, tempo e física.
- Recuperação é a substituição de um estado de erro de um processo por um estado livre de erro, podendo ser feita de duas formas: retroativa e para frente.



1 Um sistema distribuído deve incorporar muitas características que permitam o acesso seguro e contínuo, pelas entidades presentes no ambiente, aos seus recursos. Esses recursos manipulam dados que devem ser preservados, confiáveis e acessíveis dentro de parâmetros aceitáveis de desempenho e atributos como replicação e consistência conferem ao sistema essas características. Dessa forma, avalie as sentenças relacionadas aos conceitos de replicação e consistência classificando com V as sentenças verdadeiras e F as falsas:

- () Utilizar visões de dados em dispositivos pertencentes a um sistema distribuído, característica de consistência, aumentam a capacidade do sistema a responder a possíveis falhas, contribuindo também para o seu desempenho e confiabilidade.
- () Utilizar visões de dados em dispositivos pertencentes a um sistema distribuído, característica de replicação, aumentam a capacidade do sistema a responder a possíveis falhas, contribuindo também para o seu desempenho e confiabilidade.
- () Ao se utilizar os serviços de armazenamento, que permitem o compartilhamento de arquivos de máquinas locais em servidores centrais disponibilizados por uma empresa, está-se usando uma característica de replicação de dados.
- () Ao se utilizar os serviços de armazenamento, que permitem o compartilhamento de arquivos de máquinas locais em servidores centrais disponibilizados por uma empresa, está-se usando uma característica de replicação de dados. Essa característica favorece o conceito de confiabilidade que deve existir nos ambientes distribuídos.

Agora, assinale a alternativa CORRETA:

- a) () V – V – V – F.
- b) () V – F – F – V.
- c) () F – V – V – V.
- d) () V – F – V – F.

2 Talvez uma das características mais básicas de qualquer tipo de sistema é a de que ele é suscetível a uma falha, ocasionada por um erro de programação, por uma queda de conectividade, a indisponibilidade de um recurso etc. Todos esses exemplos ganham uma magnitude maior quando adicionada a eles a característica distribuída. Sendo assim, outro elemento de significativa relevância para estudo é como construir sistemas distribuídos tolerantes a falhas. Sabendo disso, avalie a sentença de afirmação e a sentença de explicação relacionadas ao conceito de tolerância a falhas envolvidas na construção de sistemas distribuídos.

- I- Um reproduutor de áudio contém uma lista de rádios que podem ser ouvidas através da internet, porém ao clicar na rádio “Classical FM” a mensagem “a lista Classical FM não pode ser usada porque não foi encontrada no repositório do servidor” é apresentada na tela do dispositivo em questão. Isso representa uma falha que precisa ser tratada por um sistema distribuído.

PORQUE

- II- O erro que ocasionou a falha precisa ser tratado pela característica de tolerância a falhas de um sistema distribuído, isto é, possibilitar que o sistema possa identificar e assumir um comportamento tolerante ao erro, mantendo a continuidade de seus serviços.

Agora, assinale a alternativa que apresenta a resposta CORRETA:

- a) () As sentenças I e II representam proposições verdadeiras, mas a II não é uma justificativa correta da I.
- b) () As sentenças I e II não representam proposições verdadeiras.
- c) () A sentença I é uma proposição verdadeira, porém a II é uma proposição falsa.
- d) () As sentenças I e II representam proposições verdadeiras, e a II é uma justificativa correta da I.

- 3 Um sistema distribuído deve incorporar muitas características que permitam o acesso seguro e contínuo, pelas entidades presentes no ambiente, aos seus recursos. Esses recursos manipulam dados que devem ser preservados, confiáveis e acessíveis dentro de parâmetros aceitáveis de desempenho e atributos como replicação e consistência conferem ao sistema essas características. Com isso, avalie as sentenças relacionadas aos conceitos de replicação e consistência:

- I- A consistência impõe uma condição a uma replicação, ou replicações, de que para ela ser (ou serem) bem sucedidas implica que as visões de dados (cópias) geradas, não necessariamente, devam possuir os mesmos estados. Se elas possuírem um estado bem próximo do original já é suficiente para garantir a característica de consistência.
- II- A consistência impõe uma condição a uma replicação, ou replicações, de que para ela ser (ou serem) bem sucedidas implica que suas visões de dados (cópias) geradas devem possuir os mesmos estados e assim garantir a característica de consistência.
- III- Os modelos de consistência lidam com as regras que as réplicas podem, ou não, obedecer para a geração de seus estados. As regras, de maneira geral, são inflexíveis em relação a diferença entre os estados dos dados originais e daqueles replicados, para garantir assim o nível de consistência desejado.
- IV- Os modelos de consistência atuam nos tipos de regras (restrições) que as réplicas devem obedecer para a geração de seus estados. Essas regras podem ser mais, ou menos, rígidas em relação a diferença entre os estados dos dados originais e daqueles replicados, garantindo assim o nível de consistência desejado.

Agora, indique a alternativa CORRETA:

- a) () As sentenças II e III estão corretas.
- b) () As sentenças II e IV estão corretas.
- c) () As sentenças I e II estão corretas.
- d) () As sentenças I e IV estão corretas.

4 Talvez uma das características mais básicas de qualquer tipo de sistema é a de que ele é suscetível a uma falha, ocasionada por um erro de programação, por uma queda de conectividade, a indisponibilidade de um recurso etc. Todos esses exemplos ganham uma magnitude maior quando adicionada a eles a característica distribuída. Sabendo disso, avalie as sentenças relacionadas aos conceitos de tolerância a falhas classificando com V as sentenças verdadeiras e F as falsas.

- () Um defeito ocorre quando um sistema cumpre parcialmente aquilo que se propõem.
- () Um erro é a causa de uma falha, e, em se tratando de sistemas distribuídos, identificar o evento, ou agente, que o causou é muito importante.
- () Uma falha representa os valores dos dados de um sistema em um dado instante de sua execução.
- () Um erro representa os valores dos dados de um sistema em um dado instante de sua execução.

Agora, assinale a alternativa CORRETA:

- a) () V – V – V – F.
- b) () V – F – F – V.
- c) () F – F – V – V.
- d) () V – F – V – F.

5 Um sistema distribuído confiável é potencialmente um sistema tolerante a falhas, dessa forma, a confiabilidade deve ser um dos atributos mais desejável de um sistema distribuído. Com isso, avalie as sentenças relacionadas ao conceito de sistema confiável:

- I- Disponibilidade é a característica que um sistema apresenta de estar pronto para uso quando acessado.
- II- Confiabilidade remete a uma característica presente nos sistemas quando são usados por períodos consideráveis de tempo sem apresentar defeitos.
- III- Um sistema é manutenível se, mesmo diante de uma falha, um estado de criticidade não é alcançado.
- IV- Quando um sistema possui a facilidade em se consertar uma falha, isto é, falhas potenciais são previamente detectadas e imediatamente recuperadas, diz-se que tal sistema é seguro.

Agora, indique a alternativa CORRETA:

- a) () A sentença I está correta.
- b) () As sentenças II, III e IV estão corretas.
- c) () As sentenças I e II estão corretas.
- d) () As sentenças III e IV estão corretas.

SEGURANÇA

I INTRODUÇÃO

No tocante à segurança em ambientes distribuídos existem, basicamente, duas visões dominantes. Uma que trata dos aspectos relativos à comunicação entre usuários e processos através de um canal seguro, considerando principalmente que essa comunicação acontece em dispositivos diferentes, logo, a necessidade de existirem os chamados canais seguros relacionados à autenticação, integridade de mensagens e também a sua confidencialidade. A outra forma de perceber a segurança diz respeito à autorização que cada processo possui sobre os seus recursos associados de um dado sistema, ou sistemas, em questão (TANENBAUM; VAN STEEN, 2007).

O gerenciamento de segurança de um sistema distribuído está diretamente associado as duas visões anteriores, uma vez que elas necessitam de mecanismos para distribuição de chaves criptográficas, e para o tratamento da inclusão e remoção de usuários de um sistema. Dessa forma, um sistema computacional confiável possui atributos de disponibilidade, fidedignidade, segurança e manutenibilidade que devem estar ainda associados a aspectos de confidencialidade e integridade. Ser **confiável** significa que os dados e serviços de um sistema distribuído só são revelados a entidades autorizadas, e ser **íntegro** significa dizer que alterações no software, hardware e dados de um sistema distribuído só acontecem com a devida autorização (TANENBAUM; VAN STEEN, 2007). Logo, para garantir a segurança de sistemas distribuídos temos que estabelecer uma política de segurança com a utilização de mecanismos.

Uma política de segurança descreve com exatidão quais ações as entidades de um sistema têm permissão de realizar e quais são proibidas. Entidades incluem usuários, serviços, dados, máquinas etc. Uma vez estabelecida a política de segurança, torna-se possível focalizar os mecanismos de segurança pelos quais a política pode ser imposta. Importantes mecanismos de segurança são: Criptografia, Autenticação, Autorização e Auditoria (TANENBAUM; VAN STEEN 2007, p. 229).

Vamos então esclarecer alguns conceitos sobre os mecanismos de segurança (STALLINGS, 2015):

- **Criptografia:** constitui uma área de estudos das variadas maneiras de encriptação, e tais maneiras constituem o que se designa por sistema criptográfico ou cifra. O processo que consiste em transformar um texto claro (*plaintext*), que representa uma mensagem original, em um texto cifrado (*ciphertext*), mensagem original codificada, é conhecido como cifração ou encriptação. O seu processo inverso, transformação da mensagem cifrada na mensagem em texto claro, é denominado decifração ou decriptação.
- **Autenticação:** trata-se de um serviço relacionado à garantia de uma comunicação autêntica. Ele pode ser visto sob dois aspectos: mensagem única (advertência ou um alarme) e de uma interação em curso (como a conexão de um cliente a um servidor). No primeiro caso esse serviço deve garantir ao destinatário que a mensagem provém da origem que ela afirma, já no segundo caso, no início da conexão o serviço garante que as entidades são autênticas e durante a conexão deve garantir que uma terceira entidade não interfira nesse processo passando por uma das entidades autenticadas.
- **Autorização:** serviço que concede um acesso a um serviço (ou recurso específico) tendo como liberação a realização de uma autenticação.

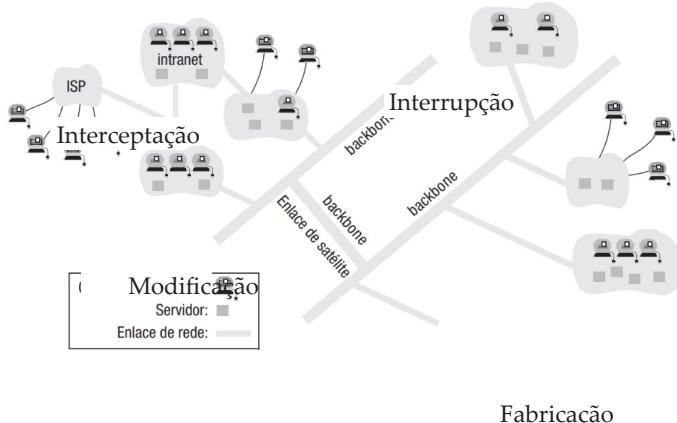
Tanenbaum e Van Steen (2007) acrescentam à lista anterior o aspecto da **Auditoria** que adiciona a perspectiva de prevenção, uma vez que possui um caráter investigativo no tocante a falhas de segurança e tomada de providência contra potenciais invasores.

Uma vez compreendidos os mecanismos de segurança, é importante que entenda a que tipos de ameaças a segurança de um sistema está sujeita (PFLEEGER; PFLEEGÉR, 2003, p. 7):

- **Interceptação:** significa que alguma parte não autorizada (pessoa, programa ou um sistema de computação) ganhou acesso a um determinado recurso. Exemplos dessa falha são: cópias ilícitas de programas ou arquivos de dados, ou programas maliciosos para obter dados na rede.
- **Interrupção:** um recurso (serviço) do sistema torna-se indisponível, é perdido, ou acessível, mas não utilizável. Exemplos dessa falha são: destruição maliciosa de um dispositivo de hardware, exclusão de um programa ou de arquivos de dados.
- **Modificação:** ocorre quando um recurso (serviço) não só é acessado, mas também adulterado. Exemplos dessa falha são: alteração dos dados de um banco de dados, alterar um programa para realizar um cálculo adicional, ou um cálculo a menos, alterar dados que estão sendo transmitidos eletronicamente.
- **Fabricação:** significa a criação de objetos falsificados em um sistema computacional. Exemplos dessa falha são: inserção de transações espúrias no sistema de comunicação de uma rede de computadores ou a adição de registros em um banco de dados.

A Figura 7 apresenta uma metáfora utilizando objetos e ações do nosso dia a dia para exemplificar os quatro tipos de ameaças listados.

FIGURA 7 – METÁFORA PARA AS AMEAÇAS AOS SISTEMAS COMPUTACIONAIS



FONTE: Pfleeger e Pfleeger (2003, p. 7)

A partir do exposto até aqui, podemos observar que a segurança pode ser resumida em permitir o acesso a alguma informação de sistemas, e aos recursos envolvidos, somente àqueles que estejam autorizados ao seu acesso (COULOURIS *et al.*, 2013), indicando ainda três classes de ameaça à segurança das informações (que complementam Pfleeger e Pfleeger (2003) apresentado anteriormente):

- **Vazamento:** a aquisição de informações por destinatários não autorizados.
- **Falsificação:** a alteração não autorizada da informação.
- **Vandalismo:** a interferência na operação correta de um sistema, sem ganho para o invasor.

A quebra de segurança, em um ambiente distribuído, por um usuário não autorizado (invasor) está associada ao seu controle sobre os canais de comunicação (lembre-se do Tópico 1 que trata exatamente desse assunto) desse ambiente. O invasor toma posse de um canal, ou ele cria um canal falso capaz de ser mascarado como um canal confiável. Dessa forma, existem métodos de ataque que podem ser utilizados por um invasor (COULOURIS *et al.*, 2013):

- **Intromissão:** mensagens são copiadas sem autorização.
- **Mascaramento:** mensagens são enviadas, ou recebidas, utilizando-se uma identidade válida, porém de uso não autorizado pelo seu portador.
- **Falsificação de mensagem:** mensagens são interceptadas, alteradas e então repassadas para o destinatário final. Essa interceptação intermediária levou ao termo *man in the middle* (o homem no meio).
- **Reprodução:** mensagens são interceptadas, armazenadas e então enviadas posteriormente em nova data. O ataque pode comprometer mesmo mensagens autenticadas e codificadas.
- **Negação de serviço:** um canal, ou outro recurso, é estressado com mensagens impedindo (negando) o acesso a outros usuários. É bastante utilizado o termo em inglês, *denial of service*.

2 TÉCNICAS DE SEGURANÇA

Depois que você aprendeu sobre os conceitos iniciais relativos à segurança, seus mecanismos, e as ameaças e métodos de ataques que podem ser utilizados em um ambiente distribuído é importante que você conheça e entenda como proteger as informações e os recursos envolvidos nesse ambiente. E, para iniciar, tomemos o Quadro 3 e o Quadro 4 como um ponto de referência para as explicações desta seção.

QUADRO 3 – SUPOSIÇÕES DE PIOR CASO E DIRETRIZES DE UM PROJETO DE SEGURANÇA

Suposições	Diretrizes
As interfaces são expostas	Os sistemas distribuídos são compostos de processos que oferecem serviços ou compartilham informações. Suas interfaces de comunicação são necessariamente abertas (para permitir que novos clientes as acessem) – um invasor pode enviar uma mensagem para qualquer interface.
As redes são inseguras	Por exemplo, as fontes de mensagem podem ser falsificadas – as mensagens podem parecer terem sido enviadas por Alice, quando na verdade foram enviadas por Mallory. Pode haver <i>spoofing</i> dos endereços dos computadores – Mallory pode se conectar na rede com o mesmo endereço de Alice e receber cópias das mensagens destinadas a ela.
Limite do tempo de vida e escopo de cada segredo	Quando uma chave secreta é gerada pela primeira vez, podemos ter certeza de que ela não foi comprometida. Quanto mais a utilizamos e mais ela se torna conhecida, maior é o risco. O uso de segredos, como senhas e chaves secretas compartilhadas, deve ter um período de validade (limite do tempo de vida) e o compartilhamento deve ser restrito.
Algoritmos e código de programa estão disponíveis para os invasores	Quanto maior e mais amplamente distribuído é um segredo, maior é o risco de sua exposição. Os algoritmos de criptografia secretos são totalmente inadequados para os ambientes atuais de redes de larga escala. A melhor prática é publicar os algoritmos usados para criptografia e autenticação, e contar com o segredo das chaves de criptografia. Isso ajuda a garantir que os algoritmos sejam poderosos, lançando-os abertamente para o escrutínio de outras pessoas.
Os invasores podem ter acesso a recursos importantes	O custo do poder de computação está diminuindo rapidamente. Devemos supor que os invasores terão acesso aos maiores e mais poderosos computadores que serão projetados durante o ciclo de vida de um sistema e, então, acrescentar algumas ordens de grandeza para admitir evoluções inesperadas.
Minimização da base confiável	As partes de um sistema que são responsáveis pela implementação de sua segurança e de todos os componentes de hardware e software com os quais elas contam, precisam ser confiáveis – isso é frequentemente referido como base de computação confiável. Qualquer defeito ou erro de programação nessa base confiável pode produzir deficiências de segurança; portanto, devemos ter como objetivo minimizar seu tamanho. Por exemplo, não se deve confiar em programas aplicativos para proteger dados de seus usuários.

FONTE: Adaptado de Coulouris et al. (2013, p. 472)



O termo *spoofing* está relacionado ao roubo de identidade. Pode ser exemplificado a partir de um cenário em que um cliente recebe um resultado diferente daquele originalmente esperado, como, por exemplo, uma mensagem falsa de e-mail (COULOURIS *et al.*, 2013).

QUADRO 4 – NOTAÇÕES CRIPTOGRÁFICAS

Notação	Significado
K_J	Chave secreta de Julia.
K_P	Chave secreta de Paulo.
K_{JP}	Chave secreta compartilhada entre Julia e Paulo.
K_{Jpriv}	Chave privada de Julia (conhecida apenas por Julia).
K_{Jpub}	Chave pública de Julia (publicada por ela para todos usarem).
$\{ALGO\}_K$	Alguma mensagem cifrada com uma chave K.

FONTE: Adaptado de Coulouris *et al.* (2013, p. 473)

Com as suposições e notações assumidas no Quadro 3 e no Quadro 4, você será apresentado a algumas técnicas utilizadas para mitigar, ou mesmo eliminar, os efeitos das condições descritas (COULOURIS *et al.*, 2013):

1- Criptografia: como apresentado anteriormente, trata-se de um processo de transformar uma mensagem original (texto claro) em um texto cifrado. A literatura pesquisada apresenta dois grupos de algoritmos usados na criptografia: um que dispõe de chaves secretas compartilhadas entre o remetente e o destinatário e que são apenas de conhecimento de ambos, e o outro grupo que dispõe de pares de chaves pública/privada. A pública disponibilizada pelo destinatário e usada pelo remetente na cifra da mensagem. E a privada é usada pelo destinatário para decifrar uma mensagem enviada.

Vejamos dois cenários que nos ajudarão a entender o Quadro 3 e o Quadro 4 (COULOURIS *et al.*, 2013):

a) Cenário 1 – comunicação autenticada com um servidor: em um ambiente corporativo existe um servidor S responsável por distribuir senhas para os usuários e que contém as chaves secretas atuais de todos os que necessitam acessar recursos do servidor, como é o caso de Julia (K_J) e Paulo (K_P). Dessa forma temos:

- I- Julia envia uma mensagem (não cifrada) para S, informando sua identidade e solicitando acesso à Paulo.
- II- S envia uma resposta para Julia, cifrada com K_j . Essa resposta será então enviada à Paulo cifrada com K_p . Porém uma nova chave secreta K_{jp} será criada para uso nesta comunicação com Paulo. Assim, a resposta recebida por Julia será algo organizado da seguinte forma: $\{\text{Resposta}\}_{K_p} K_{jp} \} K_j$, ou seja, a resposta recebida por Julia será cifrada por K_p e quando enviada a Paulo será cifrada com K_p e K_{jp} .
- III- Julia decifra a resposta usando K_j . Julia não pode decifrar nem falsificar a resposta, pois ela está cifrada por K_p . Se o destinatário não for Julia, então ele não saberá como decifrar a resposta e as mensagens estarão protegidas.
- IV- A chave K_{jp} é a chave compartilhada (chave de sessão), gerada por S, e utilizada nas interações entre Julia e Paulo.



"Esse cenário é uma versão simplificada do protocolo de autenticação originalmente desenvolvido por Roger Needham e Michael Schroeder [1978] e usado subsequentemente no sistema Kerberos, desenvolvido e usado no MIT" (STEINER et al., 1988 apud COULOURIS et al., 2013).

- a) Cenário 2 – comunicação autenticada com chaves públicas:** Paulo gerou um par de chaves pública/privada e o seguinte diálogo permite que ele e Julia estabeleçam uma chave secreta compartilhada K_{jp} . Dessa forma, temos:
- I- Paulo acessa um serviço de distribuição de chaves para obter um certificado de chave pública, fornecendo a chave pública de Julia. Após verificar a assinatura, ele lê a chave pública K_{jpub} de Julia no certificado.



Um certificado é chamado desta forma porque é assinado por uma autoridade confiável (pessoa ou organização amplamente reconhecida como sendo confiável) (COULOURIS et al., 2013). Podemos citar como exemplos instituições como: SERPRO (Serviço Federal de Processamento de Dados), Caixa Econômica Federal, Receita Federal Do Brasil, entre outros.

- II- Paulo cria uma nova chave compartilhada chamada K_{JP} . Ela é então cifrada usando K_{Jpub} com um algoritmo de chave pública. Em seguida, ele envia o resultado cifrado de K_{JP} para Julia. A este resultado cifrado é combinado um nome (*keyname*) que identifica exclusivamente um par de chaves pública/privada entre Julia e Paulo, já que ela pode ter pares de chaves enviados por outras pessoas.
- III- Julia seleciona sua chave privada correspondente (no caso K_{Jpriv}) em seu conjunto de chaves privadas e a utiliza para decifrar a mensagem e obter a chave compartilhada por Paulo (no caso K_{JP}).



O cenário anterior ilustra o uso da criptografia de chave pública para distribuir uma chave secreta compartilhada. Essa técnica é conhecida como protocolo de criptografia misto e é muito utilizada, pois explora recursos úteis tanto dos algoritmos de criptografia de chave pública como os de chave secreta (COULOURIS *et al.*, 2013, p. 476).

c) **Certificados:** “um certificado digital é um documento contendo uma declaração (normalmente curta) assinada por um principal”, ou seja, uma autoridade confiável, conforme você viu acima (COULOURIS *et al.*, 2013, p. 477). Veja então o seguinte cenário (adaptado de COULOURIS *et al.*, 2013):

- *CooperMarket* é um banco que oferece serviços através da internet aos seus associados. Logo, quando eles requisitarem algum serviço, precisam ter certeza de que estão interagindo com o *CooperMarket*. Da mesma maneira, o banco precisa autenticar seus clientes antes de liberar o acesso aos seus serviços:
 - I- Julia quer realizar compras com a loja da vendedora Regina usando um certificado do *CooperMarket*. Dessa forma, Regina pode cobrar os produtos adquiridos na conta de Julia.
 - II- O certificado de Julia é assinado usando K_{CMpriv} do banco. E Regina precisa da K_{CMpub} do banco para autenticar o certificado de Julia.
 - III- Um hipotético principal chamado Federação dos Bancos é uma entidade com autoridade conhecida e confiável que certifica e emite um certificado de chave pública para os bancos.

Ao final, teríamos algo do tipo (Quadro 5):

QUADRO 5 – CERTIFICADO DE CHAVE PÚBLICA DO COOPERMARKET

Atributos do Certificado	Valores do Certificado
1. Tipo de Certificado	Chave pública.
2. Nome	Banco CooperMarket
3. Chave Pública	$K_{CM_{pub}}$
4. Autoridade Certificadora	Federação dos Bancos
5. Assinatura	{Digest (campo1 + campo2)} $K_{FedBancopriv}$

FONTE: Adaptado de Coulouris et al. (2013, p. 478)



Funções de resumo (*digest*): as funções de resumo também são chamadas de *funções de hashing seguras* e denotadas como $H(M)$. Elas devem ser cuidadosamente projetadas para garantir que $H(M)$ seja diferente de $H(M')$ para todos os prováveis pares de mensagens M e M' . Se houver quaisquer pares de mensagens M e M' diferentes, tal que $H(M) = H(M')$, então um principal fraudulento poderia enviar uma cópia assinada de M , mas quando confrontada com ela, reivindicar que M' foi originalmente enviada e que ela deve ter sido alterada em trânsito (COULOURIS et al., 2013, p. 495).

d) Controle de acesso: seus conceitos base e técnicas de implementação foram definidos em obras como Lampson (1971) e Stallings (2008). Sistemas distribuídos recebem solicitações de acesso aos seus recursos na forma **<op, principal, recurso>** (op: operação solicitada; principal: identidade ou um conjunto de credenciais que identifica o solicitante ao recurso; recurso: ativo que está sendo acessado no ambiente distribuído pelo solicitante). E antes que o controle de acesso libere o recurso e verifique se o solicitante tem permissão a ele, a autenticação do solicitante precisa ser efetuada verificando a sua mensagem de requisição e suas credenciais (COULOURIS et al., 2013).

- **Domínios de proteção:** trata-se de uma parte do ambiente distribuído e que contém vários processos e recursos associados a ele. O domínio é construído na forma de **<recursos, direitos>** associados ao(s) seu(s) respectivo(s) processo(s). Um domínio pode estar associado a um principal que, ao ser conectado a um ambiente distribuído, após uma prévia autenticação, cria um ambiente particular para o conjunto de processos que ele pode executar. Existe um casamento entre os sistemas distribuídos e os SO, como você já bem sabe, e ela define uma relação na qual um domínio contém todos os direitos de acesso de um usuário, incluindo os adquiridos por meio de sua participação como membro de vários grupos, e que como você percebeu, é função de um SO. Então, esses direitos podem ser utilizados como delimitadores, ou como liberadores, de acesso a um dado recurso no domínio. Para a implementação de um domínio podem-se utilizar duas formas (COULOURIS et al., 2013):

- **Listas de capacidade:** um conjunto de capacidades (chaves de acesso com as quais operações podem ser realizadas em recursos presentes no ambiente) é associado a um processo observando o domínio no qual ele está inserido. Para garantir os requisitos de segurança de um ambiente distribuído ela deve ser da forma representada no Quadro 6. Dessa forma, um recurso só será acessado por clientes autenticados como partes do domínio de proteção pretendido. As requisições desses clientes obedecem a forma $\langle op, userid, capacidade \rangle$ que incluem uma capacidade para o recurso acessado, e não apenas um simples identificador, o que fornece a um servidor uma prova imediata da autorização desse cliente para acessar o recurso associado à capacidade e com as operações especificadas por ela.

QUADRO 6 – FORMATO DA LISTA DE CAPACIDADES

Atributos	Descrição
1. Identificador de Recurso	Um identificador exclusivo para o recurso pretendido
2. Operações	Uma lista das operações permitidas no recurso
3. Código de Autenticação	Uma assinatura digital tornando a capacidade impossível de ser falsificada

FONTE: Adaptado de Couloris et al. (2013, p. 480)

- I- **Listas de controle de acesso:** aos recursos estão associados à sua lista de entradas $\langle domínio, operações \rangle$, que apresentam as operações permitidas em domínio com acesso ao recurso. Já as requisições dos clientes assumem a forma $\langle op, principal, recurso \rangle$, mostrada anteriormente, para a qual o servidor precisa autenticar o principal e verificar se a operação está incluída na lista de entrada do recurso.
- II- **Implementações:** as bases de implementação para um controle de acesso seguro são as assinaturas digitais, credenciais e certificados de chave pública. Somando-se a elas, os canais seguros oferecem vantagens de desempenho, pois permitem que vários pedidos sejam executados sem verificação contínua dos principais e suas credenciais (WOBBER et al., 1994 apud COULOURIS et al., 2013).

As tecnologias como CORBA (*Common Object Request Broker Architecture*) e Java fornecem API (*Application Programming Interface*) de segurança. Elas oferecem objetos distribuídos que gerenciam seu próprio controle de acesso, métodos para autenticação e suporte para certificados, além da validação de assinaturas e suporte a criptografia de chave secreta e de chave pública. Sendo complementadas com comunicação segura, autenticação, e controle de acesso com credenciais (COULOURIS et al., 2013).

3 SEGURANÇA NOS CANAIS DE COMUNICAÇÃO

No Tópico 1 você estudou como a comunicação é vital nos ambientes de sistemas distribuídos, como ela acontece em estruturas de redes heterogêneas, e como essa heterogeneidade implica na tecnologia de rede que será utilizada, e que essa comunicação é realizada através de canais abertos entre os processos clientes e processos servidores existentes. Já no início do deste Tópico 3 você aprendeu que a segurança da comunicação entre processos pode ser alcançada através dos canais seguros (envolvendo a autenticação do canal e a integridade e confidencialidade das mensagens do canal). Logo, um canal seguro é garantido através da utilização dos mecanismos de segurança e técnicas de segurança aplicáveis aos ambientes distribuídos.

Desta forma, um canal seguro perpassa pelo estabelecimento de (TANENBAUM; VAN STEEN, 2007):

- 1- Autenticação:** um modelo de segurança está pautado na existência de uma autenticação das partes comunicantes, e que, uma vez autenticados, precisam ter também uma autorização para acessar os recursos desejados. Um processo de autenticação pode ser realizado por uma das técnicas de segurança apresentadas, previamente, neste tópico como a comunicação autenticada com chaves públicas.
- 2- Integridade e confidencialidade de mensagens:** integridade é definida como a capacidade que as mensagens devem possuir contra fraudes ou contra obtenção ilícita, enquanto a sua confidencialidade é alcançada quando não podem ser obtidas, e lidas, por intrusos. A assinatura digital é uma técnica de segurança utilizada para garantir a integridade e a confidencialidade de um canal de comunicação.
- 3- Comunicação segura entre grupos:** a complexidade dos ambientes distribuídos implica na necessidade de garantir um canal de comunicação seguro não somente entre duas partes, mas entre várias, isto é, proteger a comunicação de N usuários contra tentativas de autenticação ou de acessos não autorizados. Uma comunicação segura pode ser implementada desde o compartilhamento da mesma chave secreta (forma mais vulnerável), utilização de chaves públicas e privadas pelos membros do grupo, até a existência de servidores replicados seguros que respondem às solicitações de clientes existentes no ambiente distribuído.

RESUMO DO TÓPICO 3

Neste tópico, você aprendeu que:

- A segurança em ambientes distribuídos está associada ao estabelecimento de canais de comunicação seguros entre as entidades do sistema e a autorização que essas devem ter sobre os recursos compartilhados no ambiente.
- Aspectos de segurança envolvem também conceitos de confidencialidade e integridade entre as mensagens trocadas entre as entidades de um ambiente distribuído.
- Existe diferença entre Política de Segurança e Mecanismos de Segurança.
- Criptografia é um dos mais importantes e presentes mecanismos de segurança aplicados em ambientes distribuídos.
- É muito importante conhecer as classes de ameaças à segurança dos ambientes distribuídos.
- Existem Técnicas de Segurança que podem ser aplicadas contra as ameaças à segurança dos sistemas distribuídos.
- Canais de Comunicação Seguros são essenciais e indispensáveis para o funcionamento eficiente e confiável das trocas de mensagens entre as entidades de um ambiente distribuído.



1 Um dos principais serviços que um sistema distribuído deve implementar, de maneira eficiente e eficaz, é, sem dúvida, os seus atributos de segurança. Isso deve ser alcançado através de técnicas, mecanismos e políticas que garantam uma visão segura para todos os recursos compartilhados no ambiente distribuído. Com isso em mente, avalie as sentenças relacionadas às visões de segurança classificando com V as sentenças verdadeiras e F as falsas:

- () No tocante à visão de comunicação, a segurança é alcançada utilizando uma autenticação entre as entidades que estão trocando as mensagens e tentando acessar os recursos do sistema.
- () A integridade e a confidencialidade de mensagens estão associadas à visão de autorização que uma entidade precisa possuir para usufruir de um determinado recurso.
- () A integridade e a confidencialidade de mensagens estão associadas à visão de autenticação que uma entidade precisa possuir para usufruir de um determinado recurso.
- () O estabelecimento de um canal de comunicação seguro com uma posterior validação de autorização para uma entidade pertencente a um ambiente distribuído caracterizam as duas visões apresentadas para a segurança para sistemas distribuídos.

Agora, assinale a alternativa CORRETA:

- a) () V – V – F – F.
- b) () F – V – V – F.
- c) () F – F – V – V.
- d) () V – V – F – F.

2 A segurança de um sistema distribuído pode ser alcançada através de técnicas como a assinatura digital. Ela utiliza os atributos de integridade e de confidencialidade das mensagens, e de um canal de comunicação, para garantir os atributos de segurança. Sabendo disso, avalie a sentença de afirmação e a sentença de explicação relacionadas a essas duas características de segurança envolvidas na construção de sistemas distribuídos.

I- “É importante que a segurança de recursos de informação possua, sem dispensar, características como a de confidencialidade (proteção contra exposição para pessoas não autorizadas) e integridade (proteção contra alteração ou dano)” (COULOURIS et al., 2013, p. 18).

PORQUE

II- "Em ambientes distribuídos, clientes e servidores enviam e recebem informações em mensagens por uma rede. Essa rede, que pode ser a internet, permite que um programa em um computador se comunique com um programa em outro computador, independentemente de sua localização, com inerentes riscos de segurança associados ao livre acesso a todos os recursos disponíveis" (COULOURIS et al., 2013, p. 18).

Agora, assinale a alternativa que apresenta a resposta CORRETA:

- a) () As sentenças I e II representam proposições verdadeiras, mas a II não é uma justificativa correta da I.
- b) () As sentenças I e II não representam proposições verdadeiras.
- c) () A sentença I é uma proposição verdadeira, porém a II é uma proposição falsa.
- d) () As sentenças I e II representam proposições verdadeiras, e a II é uma justificativa correta da I.

3 Um dos principais serviços que um sistema distribuído deve implementar, de maneira eficiente e eficaz, é sem dúvida os seus atributos de segurança. Isso deve ser alcançado através de técnicas, mecanismos e políticas que garantam uma visão segura para todos os recursos compartilhados no ambiente distribuído. Dessa forma, avalie as sentenças relacionadas às visões de segurança:

- I- A integridade e a confidencialidade de mensagens não estão associadas a nenhuma das visões relacionadas à segurança em sistemas distribuídos, já que tratam de aspectos complementares relacionadas às mensagens trocadas entre as entidades presentes no sistema.
- II- Garantir a integridade e a confidencialidade de mensagens, juntamente com uma posterior validação de autorização das entidades comunicantes, fazem parte das visões de segurança presentes na implementação de ambientes distribuídos.
- III- O estabelecimento de um canal de comunicação seguro com uma posterior validação da integridade e da confidencialidade de mensagens, caracterizam as duas visões apresentadas para a segurança para sistemas distribuídos.
- IV- O controle que os processos possuem sobre os seus recursos associados a um sistema distribuído está relacionado à visão de autorização presente na segurança desse sistema.

Agora, indique a alternativa CORRETA:

- a) () As sentenças I e III estão corretas.
- b) () As sentenças II e IV estão corretas.
- c) () As sentenças I e II estão corretas.
- d) () As sentenças III e IV estão corretas.

4 A segurança de um sistema distribuído pode ser alcançada através de técnicas como a assinatura digital. Ela utiliza os atributos de integridade e de confidencialidade das mensagens, e de um canal de comunicação, para garantir os atributos de segurança. Com isso em mente, avalie as sentenças relacionadas a esses atributos de segurança classificando com V as sentenças verdadeiras e com F aquelas que são falsas:

- () Um atributo de uma mensagem deve ser um encapsulamento de proteção que evite sua apresentação às entidades sem esse direito de visualização.
- () Um atributo de uma mensagem deve ser um encapsulamento de proteção que evite sua modificação, ou estrago, por entidades sem esse direito de modificação.
- () Confidencialidade e integridade de mensagens são atributos de segurança opcionais aos sistemas distribuídos, uma vez que na internet, por exemplo, um servidor pode possuir uma senha de acesso criada por algoritmos de criptografia confiáveis que, por si só, diminuiriam os riscos de acessos não autorizados aos recursos distribuídos desse servidor.
- () Confidencialidade e integridade de mensagens são atributos de segurança indispensáveis aos sistemas distribuídos, uma vez que na internet, por exemplo, mesmo que um servidor possua uma senha de acesso criada por algoritmos de criptografia confiáveis, isso por si só, não diminuiria os riscos de acessos não autorizados aos recursos distribuídos desse servidor.

Agora, assinale a alternativa CORRETA:

- a) () V – V – F – V.
- b) () F – V – F – V.
- c) () F – F – V – V.
- d) () F – F – V – F.

5 Políticas e mecanismos de segurança são elementos fundamentais para se implementar serviços de segurança em ambientes distribuídos. E, que apesar de relacionados, possuem conceitos e utilização distintos. Com isso em mente, avalie as sentenças relacionadas aos conceitos de política e mecanismos de segurança classificando com V as sentenças verdadeiras e F as falsas:

- () Uma das garantias para os requisitos de segurança de um sistema distribuídos é a criação de mecanismos de segurança viabilizados com a utilização das políticas de segurança.
- () Uma das garantias para os requisitos de segurança de um sistema distribuídos é a criação de políticas de segurança viabilizados com a utilização dos mecanismos de segurança.
- () Recursos lógicos (serviços e dados, por exemplo), usuários e recursos físicos (máquinas equipamentos em geral) são descritos por operações precisas que podem realizar e quais dessas operações podem ser opcionais. A definição dessas operações caracteriza uma política (ou políticas) de segurança.

() Recursos lógicos (serviços e dados, por exemplo), usuários e recursos físicos (máquinas equipamentos em geral) são descritos por operações precisas que podem realizar e quais dessas operações podem ser opcionais. A definição dessas operações caracteriza um mecanismo (ou mecanismos) de segurança.

Agora, assinale a alternativa CORRETA:

- a) () F – V – F – F.
- b) () V – V – F – V.
- c) () F – V – V – F.
- d) () V – F – V – F.

IMPLEMENTAÇÕES DE SISTEMAS E APLICAÇÕES DISTRIBUÍDAS

1 INTRODUÇÃO

Ao longo da Unidade 2 e da Unidade 3 você estudou os elementos relacionados à arquitetura dos sistemas distribuídos e características relacionadas à comunicação. Já neste Tópico 4 você será convidado a estudar vários paradigmas que sustentam a construção de sistemas distribuídos, ou seja, soluções completas de *middleware*.

Conforme você já estudou na Unidade 2, um *middleware* deve abstrair um nível mais alto da programação em sistemas distribuídos, promovendo desta forma operação conjunta e portabilidade (COULOURIS *et al.*, 2013).

Assim, os paradigmas apresentados ao longo deste tópico serão os sistemas distribuídos baseados em objetos, arquivos distribuídos, baseados na *web* e os sistemas distribuídos multimídia.

A fim de apresentar a você um conteúdo de forma prática e que remeta aos conhecimentos adquiridos até aqui, cada um dos paradigmas citados será descrito a você (sempre que possível) com base em parte da organização adotada por Tanenbaum e Van Steen (2007) que prevê: os princípios de arquitetura, processos, comunicação, consistência e replicação, tolerância a falhas e segurança.

2 SISTEMAS DISTRIBUÍDOS BASEADOS EM OBJETOS

Sistemas distribuídos baseados em objetos tem como base essencial o papel do objeto: “[...] tudo é tratado como objeto, e serviços e recursos são oferecidos a clientes na forma de objetos [...]” (TANENBAUM; VAN STEEN, 2007, p. 268).

Você já deve ter ouvido falar no paradigma de orientação a objetos empregado no desenvolvimento de sistemas. Pois bem, um sistema distribuído baseado em objetos tem como principal característica a adoção do referido paradigma no seu desenvolvimento. Com isso, as entidades do sistema que se comunicam são tratadas como objetos.

Em termos gerais, a orientação a objetos auxilia a ocultar a complexidade da programação distribuída (COULOURIS *et al.*, 2013), a partir de características como encapsulamento e abstração de dados.

2.1 ARQUITETURA

Para compreendermos os princípios dos sistemas distribuídos baseados em objetos precisamos compreender que sua arquitetura está baseada no que chamamos de **objetos distribuídos**.

Objetos distribuídos estabelecem a separação entre as interfaces e os objetos que as implementam. Essa separação nos permite colocar uma interface em uma máquina e o objeto em si em outra (TANENBAUM; VAN STEEN, 2007).

Perceba que a definição de objetos distribuídos remonta os conceitos relacionados a objeto, encapsulamento e abstração. Vejamos então (COULOURIS *et al.*, 2013):

- Um **objeto** encapsula dados, também chamados **estados**, e as operações executadas neles, chamadas **métodos** (aqui estamos explicitamente falando de encapsulamento).
- Um **método** é disponibilizado por meio de uma **interface** (isso remete à abstração de dados, que separa o objeto de sua implementação).

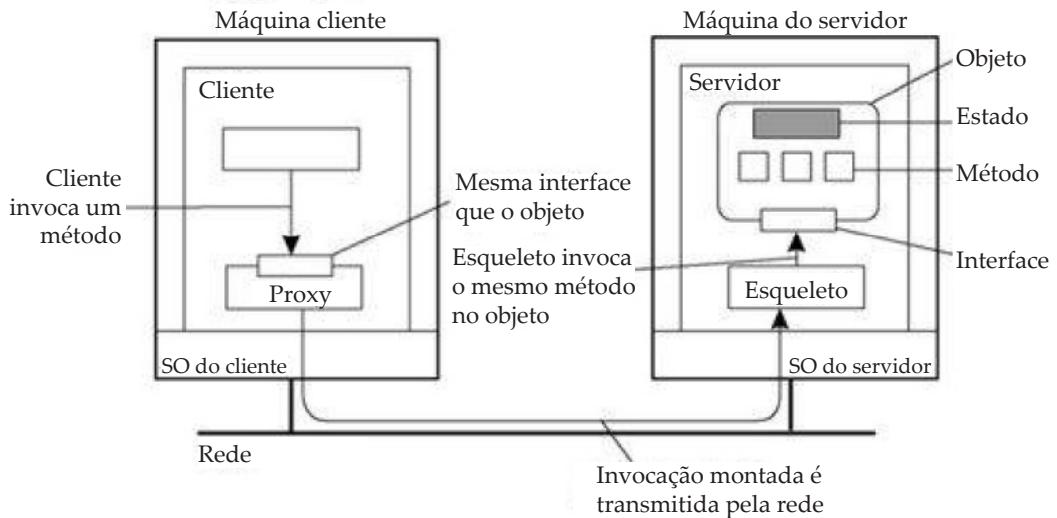
Podemos então exemplificar um objeto distribuído da seguinte forma: “quando um cliente se vincula a um objeto distribuído, uma implementação da interface do objeto, denominada *proxy*, é carregada [...]. O objeto propriamente dito reside em uma máquina do servidor, onde oferece a mesma interface que oferece na máquina cliente” (TANENBAUM; VAN STEEN, 2007, p. 268).



Um *proxy* é o equivalente, em RPC, ao *stub* do cliente. No lado servidor, o termo que denomina o equivalente ao *stub* do servidor é esqueleto.

A Figura 8 ilustra o funcionamento do objeto distribuído.

FIGURA 8 – OBJETO DISTRIBUÍDO



FONTE: Tanenbaum e Van Steen (2007, p. 269)



Em sistemas distribuídos objetos aparecem de várias formas. Há os objetos de tempo de compilação (definidos como instâncias de uma classe em nível de linguagem de programação), e os objetos de tempo de execução (utilizam adaptadores de objeto, um invólucro ao redor da implementação). Há ainda os objetos persistentes (não dependem de seu servidor corrente), e os objetos transitórios (existem somente quando seu servidor existir) (TANENBAUM; VAN STEEN, 2007).

Um bom exemplo da implementação de objetos distribuídos é o *Enterprise Java Beans* (EJB). Um EJB basicamente é definido como um objeto Java hospedado em um servidor. Os clientes remotos podem invocar o objeto de diferentes modos, uma vez que ele fornece interfaces para diferentes serviços implementados no servidor (TANENBAUM; VAN STEEN, 2007).

2.2 PROCESSOS

A partir de seus estudos sobre a arquitetura de sistemas distribuídos baseados em objetos, você já deve ter percebido que os servidores de objetos são fundamentais, afinal, eles hospedam os objetos.

Você deve estar lembrado que objetos são compostos por dados e pelo código para executar seus métodos. Logo, um servidor de objetos não fornece um serviço específico; serviços são implementados pelos seus objetos. Isso significa dizer que o servidor somente tem os meios pelos quais se pode invocar objetos locais a partir das requisições de clientes remotos (TANENBAUM; VAN STEEN, 2007).

Nesse sentido, um servidor de objetos pode invocar seus objetos de diferentes modos levando em conta, por exemplo, qual código deverá executar, sobre quais dados operar e assim por diante. Essas decisões sobre “[...] como invocar um objeto [...]” (TANENBAUM; VAN STEEN, 2007, p. 273) recebem o nome de políticas de ativação.

Exemplos de políticas de ativação poderiam considerar (TANENBAUM; VAN STEEN, 2007):

- 1- Criar um objeto transiente na primeira requisição de invocação e destruí-lo quando não houver mais clientes.
- 2- O servidor poderia colocar cada um dos seus objetos em um segmento de memória próprio. Os objetos não compartilham código nem dados.
- 3- Implementar uma única *thread* de controle.
- 4- Implementar várias *threads* de controle, uma para cada objeto.

2.3 COMUNICAÇÃO

O contexto da comunicação em sistemas distribuídos baseado em objetos está relacionado aos aspectos que acabamos de estudar sobre processos e a invocação de objetos. De maneira geral, podemos dizer que a comunicação se dá através da vinculação de um cliente a um objeto. Veja:

Quando um processo contém uma referência de objeto, em primeiro lugar ele deve se vincular ao objeto referenciado antes de invocar qualquer um de seus métodos. A vinculação resulta na colocação de um *proxy* no espaço de endereços do processo, o que implementa uma interface que contém os métodos que o processo pode invocar (TANENBAUM; VAN STEEN, 2007, p. 275).

Tratando de forma mais simplista, depois que um cliente se vincula a um processo, os métodos do objeto poderão ser invocados via *proxy*. A esse processo damos o nome de **invocação de método remoto** (*Remote Method Invocation – RMI*). Uma RMI especifica a interface de objetos em uma linguagem de definição de interfaces, por exemplo, Java. Há basicamente duas formas de se fazer uma RMI (TANENBAUM; VAN STEEN, 2007, p. 277):

- 1- **Invocação estática:** “[...] requerem que as interfaces de um objeto sejam conhecidas quando a aplicação do cliente está em desenvolvimento”.
- 2- **Invocação dinâmica:** “[...] uma aplicação seleciona qual método invocará em tempo de execução”.

2.4 CONSISTÊNCIA E REPLICAÇÃO

A literatura nos diz que sistemas distribuídos baseados em objetos seguem uma abordagem tradicional no tocante a consistência e replicação.

De maneira geral, a consistência para objetos distribuídos está associada à garantia de que, durante a invocação de objetos o acesso a eles será serializado (conforme discutimos anteriormente no Tópico 2), mantendo-os consistentes. Para tanto, é preciso impedir a execução concorrente de várias invocações ao mesmo objeto, e também garantir que duas invocações de métodos nunca ocorrerão em réplicas diferentes, simultaneamente (TANENBAUM; VAN STEEN, 2007).

Já no tocante à replicação, especial atenção deve ser dada às invocações replicadas. Consideram-se basicamente duas soluções: (1) quando se está considerando o desempenho do sistema, as replicações devem ser impedidas (MAASSEN *et al.*, 2001 apud TANENBAUM; VAN STEEN, 2007); (2) quando se está considerando a tolerância a falhas, pode-se recorrer a uma abordagem baseada em comunicação *multicast* a fim de impedir que a mesma mensagem seja enviada por réplicas diferentes.

2.5 TOLERÂNCIA A FALHAS

A exemplo da replicação, os mecanismos de tolerância a falhas aplicáveis aos sistemas distribuídos também se aplicam aos sistemas distribuídos baseados em objetos. Ainda assim, apresentaremos aqui um exemplo de como CORBA trabalha a tolerância a falhas (TANENBAUM; VAN STEEN, 2007):

Basicamente, CORBA replica objetos nos chamados **grupos de objetos** (cópias idênticas do mesmo objeto). Um grupo pode ser referenciado como se fosse um objeto único e também oferece a mesma interface que suas réplicas. Tudo isso transparente aos clientes.

2.6 SEGURANÇA

A segurança para sistemas baseados em objetos considera a proteção do objeto remoto por meio de mecanismos que padronizam a autenticação e a automatização (TANENBAUM; VAN STEEN, 2007).

3 SISTEMAS DE ARQUIVOS DISTRIBUÍDOS

Sistemas de arquivos distribuídos permitem que processos distribuídos armazenem e acessem dados remotos como se fossem locais. Isso pode ocorrer de forma segura a partir de qualquer dispositivo em uma rede (COULOURIS *et al.*, 2013; TANENBAUM; VAN STEEN, 2007).

“Os sistemas de arquivos foram originalmente desenvolvidos para os sistemas de computadores centralizados e computadores desktop [...]” (COULOURIS *et al.*, 2013, p. 522). Com o passar do tempo uma série de características relacionadas a controle de acesso e proteção foram sendo agregadas, tornando-os úteis para o compartilhamento de dados. E, como nós bem sabemos, compartilhar dados é fundamental em se tratando de sistemas distribuídos. Daí a importância de seu estudo (COULOURIS *et al.*, 2013; TANENBAUM; VAN STEEN, 2007).

3.1 ARQUITETURA

De maneira geral, os sistemas de arquivos distribuídos seguem uma arquitetura cliente/servidor, sendo a arquitetura Sistema de Arquivo de Rede (*Network File System – NFS*) da *Sun Microsystem* a mais utilizada (COULOURIS *et al.*, 2013; TANENBAUM; VAN STEEN, 2007).



A arquitetura NFS foi introduzida em 1985, tendo sido o primeiro serviço de arquivos projetado como produto. Suporta vários fornecedores. Cabe ainda mencionar que o protocolo NFSv3 (versão 3) é um padrão internet (RFC 1813) (COULOURIS *et al.*, 2013).

NFS possibilita às máquinas clientes acesso transparente a arquivos remotos, isso porque cada servidor disponibiliza uma visão padronizada de seu sistema de arquivos local (COULOURIS *et al.*, 2013; TANENBAUM; VAN STEEN, 2007).

O acesso transparente por parte dos clientes a arquivos remotos é possível graças a um protocolo de comunicação que permite que um conjunto de diferentes processos, executados em máquinas com diferentes SO, compartilhe um sistema de arquivos em comum (COULOURIS *et al.*, 2013; TANENBAUM; VAN STEEN, 2007).

Em complemento, no NFS, cada computador pode ser tanto cliente quanto servidor, ou seja, os arquivos de qualquer máquina podem ser compartilhados (COULOURIS *et al.*, 2013), o que também justifica sua independência de hardware e SO.

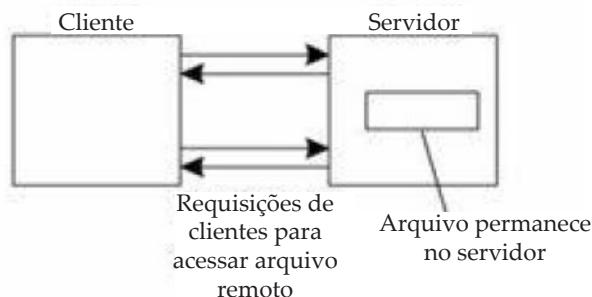
Mas, poderíamos então perguntar, como ocorrem esses processos de compartilhamento de arquivos distribuídos? A resposta está no que denominamos **serviço de arquivo remoto**, que permite acesso transparente dos clientes a um sistema de arquivos de um servidor remoto (TANENBAUM; VAN STEEN, 2007).

“Os sistemas de arquivos são responsáveis pela organização, armazenamento, recuperação, atribuição de nomes, compartilhamento e proteção de arquivos” (COULOURIS *et al.*, 2013, p. 525).

Assim, esse acesso se dá a partir de dois modelos (TANENBAUM; VAN STEEN, 2007):

- **Modelo de acesso remoto:** o cliente tem acesso a uma interface com as operações sobre os arquivos, mas as operações são de fato implementadas no servidor, conforme Figura 9.

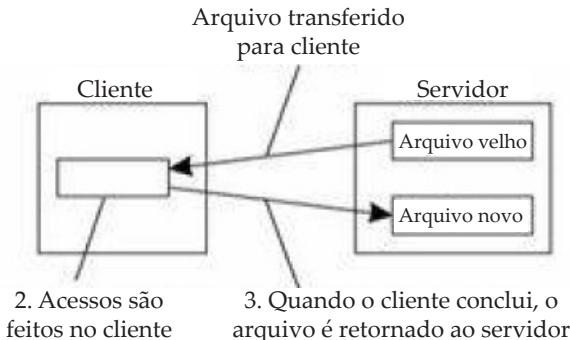
FIGURA 9 – MODELO DE ACESSO REMOTO



FONTE: Adaptado de Tanenbaum e Van Steen (2007, p. 296)

- **Modelo de carga/atualização:** o cliente baixa e acessa os arquivos localmente, subindo o arquivo de volta ao servidor após o término de sua utilização, conforme ilustrado na Figura 10. O serviço FTP (*File Transfer Protocol*) é um exemplo desse modelo.

FIGURA 10 – MODELO DE CARGA/ATUALIZAÇÃO



FONTE: Adaptado de Tanenbaum e Van Steen (2007, p. 296)

3.2 PROCESSOS

Em se tratando de processos em sistemas de arquivos distribuídos, o tema de maior interesse gira em torno da seguinte questão: se os processos devem ou não ter estado.

Para tratar deste assunto, tomaremos como exemplo o comportamento do NFS. Nas versões 2 e 3, os servidores NFS eram sem estado, ou seja, “[...] o protocolo NFS não exigia que os servidores mantivessem nenhum estado do cliente” (TANENBAUM; VAN STEEN, 2007, p. 302).

Basicamente, a vantagem de se assumir uma abordagem como esta é a simplicidade, uma vez que, como não são mantidos os estados do cliente, caso o servidor venha a cair, não há a necessidade de o servidor retornar a um estado anterior. Contudo, sabemos que esta não é uma situação aceitável, e uma abordagem com estado é de fato mais interessante.

Assim, a partir da versão 4 o NFS abandonou a abordagem sem estado, não somente pelo que acabamos de mencionar, mas também porque deve “[...] funcionar também em redes de longa distância. Isso requer que os clientes possam fazer uso efetivo de caches, o que, por sua vez, requer um protocolo de consistência de cache eficiente” (TANENBAUM; VAN STEEN, 2007, p. 302). Em outras palavras, em um cenário como este é importante que um servidor mantenha informações sobre os arquivos de seus clientes.

3.3 COMUNICAÇÃO

No tocante à comunicação em sistemas de arquivos distribuídos, pode-se dizer que não existe nada de diferente daquilo que você já tenha estudado. De fato, em um contexto geral, a comunicação se dá através de RPC, especialmente para manter a independência de SO, rede e protocolos (TANENBAUM; VAN STEEN, 2007).

Para retomarmos aqui o funcionamento de uma RPC agora em um sistema de arquivos distribuídos, tomaremos como base novamente o NFS. No NFS, uma operação pode “[...] ser implementada como uma única chamada de procedimento remoto a um servidor de arquivos” (TANENBAUM; VAN STEEN, 2007, p. 303). São os chamados **procedimentos compostos**. Em um procedimento composto “[...] várias RPC podem ser agrupadas em uma única requisição [...]” (TANENBAUM; VAN STEEN, 2007, p. 303).

Um aspecto de interesse a ser considerado em relação aos procedimentos compostos é que não existe uma “semântica transacional” associada. Por exemplo: (1) as operações que compõem um procedimento composto são executadas obedecendo a ordem de suas requisições, nada mais; (2) no caso de operações concorrentes de outros clientes, conflitos não são evitados; (3) se uma operação do procedimento composto falhar, nenhuma outra será executada (TANENBAUM; VAN STEEN, 2007).

3.4 CONSISTÊNCIA E REPLICAÇÃO

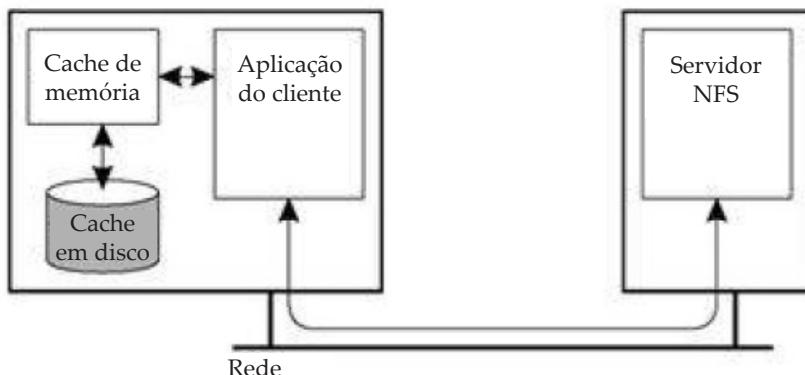
Aspectos como consistência e replicação são de grande importância no estudo de sistemas de arquivos distribuídos, sobretudo em se tratando de redes de longa distância.

Conforme você já estudou, a replicação consiste da realização de cópias de dados em diferentes locais. Da mesma forma, a replicação em sistemas de arquivos distribuídos mantém várias cópias do conteúdo do arquivo.

A replicação do lado do servidor não costuma ser uma estratégia comum. Apesar de permitir que “[...] vários servidores compartilhem a carga do fornecimento de um serviço para clientes que acessam o mesmo conjunto de arquivos [...]” (COULOURIS *et al.*, 2013, p. 528), é aplicável somente para tolerância a falhas, visando disponibilidade (escalabilidade do serviço) (TANENBAUM; VAN STEEN, 2007).

Por outro lado, de maneira geral, o serviço de cache exerce papel especial do lado do cliente. A cache de dados no lado cliente (um cliente NFS, por exemplo) funciona conforme está demonstrado na Figura 11:

FIGURA 11 – CACHE NO LADO CLIENTE NO NFS



FONTE: Adaptado de Tanenbaum e Van Steen (2007, p. 296)

No modelo descrito na Figura 10 “cada cliente pode ter uma cache de memória que contém dados lidos anteriormente em um servidor. Além disso, também pode haver uma cache em disco que é adicionada como uma extensão à cache de memória, usando os mesmos parâmetros de consistência” (TANENBAUM; VAN STEEN, 2007, p. 315).



NFSv3 implementava diferentes políticas de *cache*, contudo, não havia garantias de consistência. Os dados em *cache* poderiam ficar desatualizados por até trinta segundo sem que o cliente viesse a saber. Alguns problemas de consistência foram corrigidos no NFSv4 (TANENBAUM; VAN STEEN, 2007).

3.5 TOLERÂNCIA A FALHAS

Tolerância a falhas é essencial para que operações sobre arquivos distribuídos continuem a funcionar diante de falhas em clientes e/ou servidores (COULOURIS *et al.*, 2013).

De modo geral, é aplicada em sistemas de arquivos distribuídos conforme os aspectos que você estudou anteriormente no Tópico 2, sendo que, em muitos casos, a replicação é utilizada na criação de grupos de servidores (TANENBAUM; VAN STEEN, 2007).

Vale, no entanto, mencionar que, em se tratando de tolerância a falhas, a ocorrência de falhas arbitrárias nos servidores costuma ser ignorada. Isso (como era de se imaginar) é um problema, especialmente se estivermos falando de internet. Para situações assim, “[...] é possível construir soluções práticas considerando esgotamento de temporizações razoáveis e inicializando novos grupos de servidores [...]” (TANENBAUM; VAN STEEN, 2007, p. 328).

3.6 SEGURANÇA

Semelhante às seções anteriores muitas das questões relacionadas à segurança são aplicadas a sistemas de arquivos distribuídos.

De maneira geral, os sistemas de arquivos distribuídos cliente/servidor devem realizar a autenticação das requisições dos clientes para controlar o acesso aos servidores (COULOURIS *et al.*, 2013; TANENBAUM; VAN STEEN, 2007). O mesmo vale para o NFS que temos adotado como exemplo até agora.

Em NFS, o estabelecimento de um canal de comunicação seguro entre cliente e servidor se dá a partir de vários aspectos, merecendo destaque: as RPC seguras, o controle de acesso e a autenticação descentralizada (TANENBAUM; VAN STEEN, 2007):

- **RPC segura:** o NFSv4 oferece suporte a um ambiente de segurança denominado RPSEC_GSS. Com ele um sem número de mecanismos de segurança para prover canais seguros é suportado, além de garantir integridade e confidencialidade de mensagens.
- **Controle de acesso:** suportado pelo atributo ACL (*Access Control List*), uma lista de entradas para o controle de acesso. Cada entrada estabelece os direitos de acesso a um usuário ou grupo de usuários (tipos de usuários podem ser vistos no Quadro 7). Operações estabelecidas pelos direitos de acesso incluem leitura, escrita e execução de arquivos, entre outras.

QUADRO 7 – TIPOS DE USUÁRIO NFS

Tipo de usuário	Descrição
Proprietário	O proprietário de um arquivo
Grupo	Grupo de usuários associado com um arquivo
Todos	Qualquer usuário ou processo
Interativo	Qualquer processo que acesse o arquivo com base em um terminal interativo
Rede	Qualquer processo que acesse o arquivo por meio da rede
Lote	Qualquer processo que acesse o arquivo como parte de um <i>job</i> em lote
Anônimo	Qualquer um que acesse o arquivo sem autenticação
Autenticação	Qualquer usuário ou processo autenticado
Serviço	Qualquer processo de serviço definido

FONTE: Adaptado de Tanenbaum e Van Steen (2007, p. 324)

- **Autenticação descentralizada:** possível a partir da utilização de sistemas de arquivos seguros (*Secure File Systems – SFS*). O SFS “[...] permite que nomes de arquivos incluam informações sobre a chave pública do servidor de arquivos”, o que “[...] simplifica o gerenciamento de chaves em sistemas de grande escala” (TANENBAUM; VAN STEEN, 2007, p. 328).

4 SISTEMAS DISTRIBUÍDOS BASEADOS NA WEB

Não há como falarmos das várias aplicações práticas dos sistemas distribuídos e não considerar os sistemas baseados na web. Desde a sua criação, a *World Wide Web* (WWW), ou simplesmente web, cresceu e se expandiu rapidamente, especialmente a partir do momento que os usuários passaram a acessar interfaces gráficas (TANENBAUM; VAN STEEN, 2007).



O padrão web teve início no Laboratório Europeu de Física Nuclear (*European Particle Physics Laboratory*, CERN), em Genebra, como um projeto que permitiria a seu grupo de pesquisadores, numeroso e geograficamente disperso, acessar documentos compartilhados por meio de um sistema simples de hipertexto (TANENBAUM; VAN STEEN, 2007, p. 330).

Em decorrência de sua evolução, a web passou de sistema simples baseado em documentos, para um sistema distribuído com milhões de clientes e servidores, compartilhando também serviços (TANENBAUM; VAN STEEN, 2007).



"Desde 1994, o *World Wide Web Consortium*, uma colaboração entre o CERN e o MIT, vem trabalhando no desenvolvimento da Web. Esse consórcio é responsável por padronizar protocolos, melhorar a interoperabilidade e aprimorar as capacidades da Web" (TANENBAUM; VAN STEEN, 2007, p. 330).

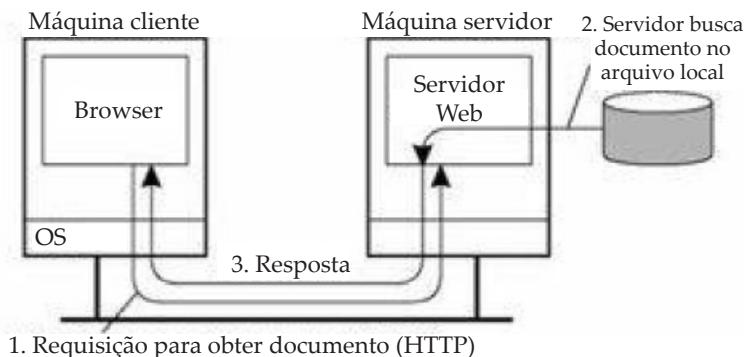
4.1 ARQUITETURA

Conforme você acabou de estudar, a evolução da web promoveu uma série de mudanças. Passou-se do suporte a documentos estáticos para distribuídos, e mais recentemente a serviços. Tais mudanças interferiram na organização de sua arquitetura.

Tradicionalmente, os sistemas baseados na web apresentam-se em uma arquitetura cliente/servidor: um cliente interage com um servidor por meio de um **navegador** (*browser*), que nada mais é do que uma aplicação especial responsável por apresentar os **documentos web** adequadamente. Um documento web, para ser apresentado adequadamente, normalmente é escrito em uma **linguagem de marcação** (como HTML, por exemplo), a qual referencia **documentos embutidos** (como um programa completo, por exemplo) (TANENBAUM; VAN STEEN, 2007).

Finalmente, cliente e servidor se comunicam através de um protocolo de transferência de hipertexto (*Hypertext Transfer Protocol – HTTP*). Veja como essa arquitetura é organizada na Figura 12:

FIGURA 12 – ORGANIZAÇÃO DA ARQUITETURA DE FUNCIONAMENTO DE UM SISTEMA WEB



FONTE: Tanenbaum e Van Steen (2007, p. 331)

Uma importante melhoria agregada a esta forma de comunicação foi o **script do lado do servidor**. Com a introdução de scripts, servidores passaram a processar um documento antes de passá-lo para o cliente, ou seja, o servidor envia ao cliente o resultado da execução do script (TANENBAUM; VAN STEEN, 2007).

Com o processamento do lado do servidor a arquitetura de sistemas baseados na web evoluiu para uma estrutura em três camadas com: um servidor web, um servidor de aplicação (responsável pela execução de programas) e um servidor de banco de dados. Infelizmente, a arquitetura baseada em três camadas trouxe consigo a redução de desempenho, em geral ocasionada por gargalos entre aplicação e banco de dados (TANENBAUM; VAN STEEN, 2007).

Outro contexto arquitetural que merece atenção em seus estudos são os **serviços web**. Por definição, um serviço web “[...] nada mais é do que um serviço tradicional (por exemplo, [...] um serviço de previsão do tempo [...]) que é oferecido pela Internet” (TANENBAUM; VAN STEEN, 2007, p. 333).

A ideia básica de utilização dos serviços web está no fornecimento de uma “[...] infraestrutura para manter uma forma mais rica e mais estruturada de interoperabilidade entre clientes e servidores” (COULOURIS *et al.*, 2013, p. 382). Em outras palavras, permitem que uma aplicação cliente possa invocar serviços fornecidos por uma aplicação no servidor.

Para tanto, a comunicação entre as máquinas cliente e servidor se utiliza do **protocolo simples de acesso a objeto** (*Simple Object Access Protocol – SOAP*), para a padronização da comunicação entre os processos. Além disso, serviços web possuem uma descrição do serviço conforme padrão **integração, descoberta e descrição universal** (*Universal Description, Discovery and Integration – UDDI*), e são descritos pela **linguagem de definição de serviços web** (*Web Services Definition Language – WSDL*).

4.2 PROCESSOS

Neste subtópico, dedicaremos atenção aos processos de maior importância para os sistemas distribuídos baseados na web: os processos do cliente e do servidor. Iniciando pelos processos do cliente, trataremos do navegador e do *proxy* (TANENBAUM; VAN STEEN, 2007):

- **Navegador web:** considerado o software mais importante, é através do navegador que o usuário “navega” pelas páginas web que lhes são apresentadas na tela. Idealmente independentes de plataforma, navegadores tem uma estrutura constituída de: (1) lógica do navegador (responsável pelos mecanismos através dos quais um usuário acessa um documento); e (2) rotinas de apresentação (contém o código para apresentar os documentos *web* adequadamente).
- **Proxy web:** “originalmente [...] usado para permitir a um browser manipular protocolos da camada de aplicação que não fossem HTTP [...]” (TANENBAUM; VAN STEEN, 2007, p. 336). Atualmente os navegadores não precisam mais de *proxies* para esta finalidade, mas por outras razões, como: filtrar requisições e respostas, comprimir arquivos, e especialmente, armazenar.

Já no tocante aos processos servidores, abordaremos brevemente aqui aquele que é considerado o servidor *web* mais popular: o Apache.

Considerado como um servidor web geral, o Apache é configurado, falando-se de maneira simplista, para produzir uma resposta a cada requisição. De fato, o Apache é “uma peça complexa de software”, com alta capacidade de configuração e extensibilidade, e também independente de plataformas específicas (TANENBAUM; VAN STEEN, 2007).

4.3 COMUNICAÇÃO

Em termos gerais, a comunicação entre sistemas distribuídos baseados na web se dá de duas formas: via HTTP para os sistemas tradicionais, e via SOAP para os sistemas baseados em serviços (TANENBAUM; VAN STEEN, 2007):

- **HTTP:** este é um protocolo cliente/servidor de funcionamento relativamente simples, no qual uma requisição é enviada de um cliente a um servidor, e a resposta deste último é retornada. Importante mencionar que sempre que uma requisição é emitida, primeiramente se estabelece uma conexão TCP (*Transmission Control Protocol*) com o servidor, e só então a mensagem é enviada. O mesmo acontece para o recebimento da resposta.
- **SOAP:** protocolo padrão para comunicação com serviços web, a maioria dos canais de comunicação SOAP é implementada por intermédio do HTTP. Seu uso leva em conta que duas partes que se comunicam têm pouco conhecimento uma sobre a outra. Por isso, sua finalidade é fornecer um meio simples para que essa comunicação aconteça de forma simples, o que justifica o fato de as mensagens SOAP serem, em sua maioria, baseadas em XML (*Extensible Markup Language*).



Apesar de a linguagem XML facilitar o uso de um interpretador geral, sua sintaxe é caracterizada pelo excesso de palavras, o que gera um gargalo de desempenho quando da interpretação de mensagens XML (TANENBAUM; VAN STEEN, 2007).

4.4 CONSISTÊNCIA E REPLICAÇÃO

Já mencionamos anteriormente que um importante aspecto relacionado aos sistemas distribuídos baseados na web é assegurar requisitos de desempenho e disponibilidade. Independentemente do suporte ao conteúdo estático, ou da crescente oferta de conteúdo dinâmico, o uso de *cache* e replicação de conteúdo tem auxiliado sobremaneira para a melhoria do desempenho em escala global (TANENBAUM; VAN STEEN, 2007).

A *cache* em sistemas distribuídos baseados na web ocorre em dois contextos: nos navegadores e na implementação de um *proxy* web. No caso dos navegadores, estes, em geral, possuem um recurso de *cache* simples. Assim, quando um usuário busca um documento, ele é armazenado em *cache*, e em uma próxima ocasião, é a partir da *cache* que será carregado (TANENBAUM; VAN STEEN, 2007).

Já o *proxy* web pode implementar uma *cache* compartilhada, ou seja, um site cliente que implementa um *proxy* web pode aceitar requisições locais e repassá-las para servidores web (TANENBAUM; VAN STEEN, 2007).

Outro tema que merece sua atenção remete à replicação para sistemas de hospedagem web. Nesse sentido, as redes de entrega de conteúdo (*Content Delivery Network* – CDN) representam um serviço de hospedagem web. Uma CDN fornece “[...] infraestrutura de distribuição e replicação de documentos Web de vários sites por toda a Internet” (TANENBAUM; VAN STEEN, 2007, p. 348), garantindo fácil acesso e disponibilidade.

Para tanto, a CDN leva em conta três aspectos para a replicação: (1) estimativa de métricas (mensurar as melhores propostas em prol do desempenho); (2) ajuste de adaptação (uma vez estabelecida uma métrica, o que fazer para se adaptar a ela); e (3) providências de ajustamento (mudar posicionamento de réplicas, mudar imposição de consistência e decidir como/quando redirecionar requisições de clientes).

4.5 TOLERÂNCIA A FALHAS

Obtém-se a tolerância a falhas em sistemas distribuídos baseados na web basicamente através de cache do lado do cliente e da replicação de servidores. Em se tratando de serviços web, a literatura nos diz que não existe nada de especial nesse sentido, apesar de a simples replicação do servidor não servir para serviços web.

O fato anterior é percebido porque “[...] serviços Web suportam transações distribuídas em redes remotas [...]” (TANENBAUM; VAN STEEN, 2007, p. 353) e há possibilidade de falha em serviços ou canais de comunicação não confiáveis, gerando alguns problemas no uso da replicação.

4.6 SEGURANÇA

Quando abordamos as questões de segurança para sistemas distribuídos baseados na web, estamos basicamente considerando a construção de um canal seguro (de vários ataques) entre cliente e servidor.

Nesse sentido, a abordagem de segurança considerada a mais usual é a utilização da **camada de soquetes seguros** (*Secure Socket Layer* – SSL), ou sua atualização (RFC 2246 e RFC 3346), **segurança na camada de transporte** (*Transport Layer Security* – TLS) (TANENBAUM; VAN STEEN, 2007).

Por definição, TLS é um protocolo de segurança, livre de aplicações, e que costuma ser baseado em TCP. Seu núcleo é formado pela camada de protocolo de registro TLS, que é a responsável pela implementação de um canal seguro entre cliente e servidor (TANENBAUM; VAN STEEN, 2007).

5 SISTEMAS MULTIMÍDIA DISTRIBUÍDOS

Estamos chegando ao término de seus estudos sobre sistemas distribuídos. Separamos para este final um assunto que acreditamos manterá ainda mais sua atenção, dados todos os recursos que temos hoje a nossa disposição.

Falar sobre aplicações multimídia distribuídas não poderia então ficar de fora, já que você está vivenciando esse tipo de sistema, inclusive aqui em sua jornada no ensino superior à distância.

Iniciaremos recordando o que comentamos no Tópico 2 desta Unidade 3, quando falamos acerca da comunicação baseada em fluxo de dados ou *stream*. A capacidade de manipulação desses fluxos baseados no tempo (áudio e vídeo, por exemplo) permitiu o desenvolvimento de sistemas multimídia distribuídos (COULOURIS *et al.*, 2013), tais como telefonia sobre internet e videoconferência, por exemplo.

Perceba que, ao definirmos sistemas multimídia, falamos em “fluxos de dados baseados no tempo”. Esta sentença nos leva a uma característica essencial dos sistemas multimídia: exigem a distribuição dos fluxos de dados em concordância com restrições temporais, ou seja, são sistemas de tempo real (COULOURIS *et al.*, 2013).

Contudo, existe uma diferença entre sistemas multimídia distribuídos e outros tipos de sistemas de tempo real. Sistemas de tempo real, como os de controle de tráfego aéreo, por exemplo, lidam com volumes de dados relativamente pequenos, contudo, o não cumprimento de prazos pode acarretar em graves consequências. Para que não haja problemas os recursos computacionais são alocados com escalonamento fixo de modo a garantir o atendimento dos requisitos do sistema (COULOURIS *et al.*, 2013).

Esta solução, no entanto, não se aplica a sistemas multimídia distribuídos já que estes precisam apresentar seus resultados (o retorno de áudio em uma conferência, por exemplo), “[...] de acordo com um escalonamento determinado externamente” (COULOURIS *et al.*, 2013, p. 882).

“As consequências do não cumprimento dos prazos finais em aplicativos multimídia podem ser sérias, especialmente em ambientes comerciais, como nos serviços sob demanda [...]” (COULOURIS *et al.*, 2013, p. 882). Um exemplo desta situação seria você estar assistindo a um episódio de sua série favorita, no dia de seu lançamento mundial, a partir de seu serviço de *streaming*, e o serviço “travar”.

Esta situação acontece porque (COULOURIS *et al.*, 2013):

- 1- Sistemas multimídia distribuídos competem com outros aplicativos distribuídos por largura de banda e, também, pelos recursos de outros dispositivos na rede.

- 2- Requisitos para o uso de recursos são dinâmicos, ou seja, quanto maior o número de usuários conectados ao mesmo tempo, no mesmo serviço de *streaming*, mais largura de banda será exigida.
- 3- Os aplicativos multimídia podem coexistir na mesma rede e até na mesma máquina.

5.1 DADOS MULTIMÍDIA

Agora que já contextualizamos os sistemas multimídia distribuídos vamos detalhar um pouco mais o que são os dados multimídia bem como suas principais características. São características gerais de dados multimídia, como áudio e vídeo, serem baseados no tempo (conforme já comentamos anteriormente) e contínuos (COULOURIS *et al.*, 2013):

- **Baseados no tempo:** áudio e vídeo são formados pelo que chamamos elementos de dados individuais (vídeos são formados por quadros, por exemplo), desta forma definem o conteúdo do fluxo. Por isso, os tempos no quais os elementos individuais são produzidos/gravados afetam a validade do fluxo de dados.
- **Contínuos:** “[...] a mídia contínua é representada como sequências de valores discretos que substituem uns aos outros com o passar do tempo” (COULOURIS *et al.*, 2013, p. 886). Em outras palavras, é a continuidade que nos permite visualizar e/ou ouvir um fluxo de dados.

Vale ainda mencionar outra característica da multimídia: seu tamanho. Sistemas distribuídos multimídia precisam ser capazes de manipular esses dados mais volumosos sem perder desempenho. O Quadro 8 mostra alguns exemplos de fluxos de dados em função de seu tamanho e frequência.

QUADRO 8 – CARACTERÍSTICAS DOS FLUXOS DE MULTIMÍDIA TÍPICOS

Fluxos de Multimídia	Taxa de dados (aproximada)	Amostra ou quadro	
		Tamanho	Frequência
Conversação telefônica	64 kbps	8 bits	8000/s
Som com qualidade de CD	1,4 Mbps	16 bits	44000/s
Vídeo de TV padrão (não compactado)	120 Mbps	Até 640 x 480 pixels x 16 bits	24/s
Vídeo de TV padrão (MPEG-1 compactado)	1,5 Mbps	Variável	24/s
Vídeo HDTV (não compactado)	1000 a 3000 Mbps	Até 1920 x 1080 pixels x 24 bits	24-60/s
Vídeo HDTV (MPEG-2/MPEG-4 compactado)	6-20 Mbps	Variável	24-60/s

FONTE: Coulouris et al. (2013, p. 886)

Repare que alguns tipos de fluxos exigem maior largura de banda que outros. Já mencionamos antes que aplicações competem entre si por largura de banda, o que compromete o desempenho delas.

Para resolver esse problema, faz-se uso da compactação dos fluxos de dados. Formatos de fluxos de vídeo compactados envolvem MPEG-1, MPEG-2 e MPEG-4, que você já deve ter tido a oportunidade de conhecer. A compactação de fluxos de dados tem como objetivo reduzir os requisitos de largura de banda, sem afetar os requisitos de temporização dos dados contínuos (COULOURIS *et al.*, 2013).

5.2 GERENCIAMENTO DE QUALIDADE DE SERVIÇO

Até aqui verificamos as características dos sistemas multimídia e dos seus tipos de dados. Você verificou que dadas as características relacionadas à continuidade e tempo, atrasos na distribuição do fluxo de dados podem ser críticos.

A capacidade de o sistema multimídia realizar suas tarefas e conseguir seus resultados é denominada **qualidade de serviço** (*Quality of Service – QoS*) (COULOURIS *et al.*, 2013). Em consequência, a capacidade de alocação dos recursos (largura de banda e memória, por exemplo) para corresponder às necessidades de qualidade chama-se **gerenciamento de qualidade de serviço**.

O gerenciamento de qualidade de serviço se mostra crítico para sistemas multimídia distribuídos porque, conforme já comentamos anteriormente nesta seção, aplicações multimídia concorrem entre si e com outras aplicações convencionais por recursos como largura de banda, processamento, armazenamento, entre outros. De maneira geral, os métodos convencionais,

[...] para compartilhamento de ciclos de processador e largura de banda de rede não podem satisfazer as necessidades das aplicações multimídia [...]. Uma distribuição atrasada não tem valor. [...] as aplicações precisam garantir que os recursos necessários sejam alocados e escalonados nos momentos exigidos (COULOURIS *et al.*, 2013, p. 888).

Como então se garante a qualidade do serviço? Basicamente, “uma requisição de qualidade do serviço bem sucedida gera uma garantia de qualidade do serviço para o aplicativo e resulta na reserva, e no subsequente escalonamento, dos recursos solicitados” (COULOURIS *et al.*, 2013, p. 881).

Para termos isso mais claro, imaginaremos o cenário de uma conferência (uma aula on-line, por exemplo). Em um ambiente como este o *stream* de mídia é contínuo, sendo processado e distribuído pelo estabelecimento de conexões entre seus elementos de dados, transferindo-os em sequência de uma origem para um

destino. Para que os elementos de dados do vídeo (seus quadros), por exemplo, cheguem ao seu destino em tempo, há uma coordenação (gerenciamento) de uma série de elementos (tempo de CPU, largura de banda, uso frequente de recursos, entre outros). Tudo isso só é possível pela existência de um componente de sistema que gerencia a qualidade do serviço. A literatura o chama, convenientemente, de gerenciador de qualidade do serviço.

5.3 GERENCIAMENTO DE RECURSOS

Seu estudo sobre sistemas multimídia distribuídos até aqui mostrou a você que estamos tratando de um tipo de sistema de tempo real, cujas aplicações distribuídas competem por recursos para entregar seus fluxos de dados em tempo.

Neste cenário, vamos, novamente, chamar sua atenção para a expressão “competem por recursos”: isso significa que uma aplicação precisa de recursos suficientes para ser executada (como você já estudou anteriormente, isso é desempenho) e também precisa que os recursos estejam disponíveis (e isso é escalonamento) (COULOURIS *et al.*, 2013).

Para que isso ocorra, é necessário que haja o gerenciamento de tais recursos, que se dá através do que a literatura chama de escalonamento de recursos. Métodos de escalonamento são aplicados a todo recurso que venha afetar o desempenho de nosso sistema multimídia distribuído, basicamente atribuindo aos recursos prioridades (COULOURIS *et al.*, 2013):

- **Escalonamento imparcial:** conforme o nome sugere, este método é imparcial ao escalar um recurso quando vários fluxos concorrem por ele. Basicamente é aplicado um escalonamento em rodízio (*round-robin*), aplicado bit a bit a todos os fluxos, proporcionando imparcialidade sobre os tamanhos variados e tempos de chegada de pacotes.
- **Escalonamento em tempo real:** há uma série de escalonamentos de tempo real criados para escalar de CPU, por exemplo, e que se adaptam ao nosso objeto de estudo aqui. Pode-se mencionar o EDF (*Earliest-Deadline-First*), no qual o elemento com prazo final mais adiantado, ou seja, com prazo final mais cedo, é processado primeiro; e o RM (*Rate-Monotonic*) no qual quanto maior a velocidade dos itens de um fluxo, maior a sua prioridade.



A Leitura Complementar selecionada para você nesta unidade traz um estudo de caso de uma aplicação de sistemas distribuídos muito conhecida por nós todos: o Google. Na sequência você verá parte do trecho introdutório retirado da obra de Coulouris *et al.* (2013) que apresenta o Google em perspectiva: enquanto mecanismo de busca e enquanto provedor de nuvem. O material completo encontra-se disponível em: <https://sdpisutic.files.wordpress.com/2017/08/sistemas-distribuc3addos-conceitos-e-projeto-2013.pdf>.

LEITURA COMPLEMENTAR

PROJETO DE SISTEMAS DISTRIBUÍDOS – ESTUDO DE CASO: GOOGLE

George Coulouris
Jean Dollimore
Tim Kindberg
Gordon Blair

O Google [www.google.com] é uma corporação dos Estados Unidos, sediada em Mountain View, Califórnia (a Googleplex), oferecendo pesquisa na Internet e aplicações web mais gerais, obtendo seus lucros principalmente com os anúncios ligados a tais serviços. O nome é um trocadilho com a palavra googol, o número 10100 (ou 1 seguido por cem zeros), enfatizando a enorme escala das informações disponíveis na Internet atualmente. A missão do Google é dominar esse enorme volume de informações: “organizar as informações do mundo e torná-las universalmente acessíveis e úteis” [www.google.com].

O Google nasceu de um projeto de pesquisa na Universidade de Stanford, com a empresa sendo inaugurada em 1998. Desde então, ela cresceu até ter uma fatia dominante do mercado da pesquisa na Internet, principalmente devido à eficiência do algoritmo de classificação subjacente utilizado em seu mecanismo de busca. De forma significativa, o Google diversificou e, além de fornecer um mecanismo de busca, agora tem uma importante participação na computação em nuvem.

Da perspectiva dos sistemas distribuídos, o Google oferece um fascinante estudo de caso, com requisitos extremamente exigentes, particularmente em termos de escalabilidade, confiabilidade, desempenho e abertura. Por exemplo, em termos de pesquisa, vale notar que o sistema subjacente tem se adaptado muito bem ao crescimento da empresa. Desde seu sistema de produção inicial (em 1998), até lidar com mais de 88 bilhões de pesquisas por mês (em 2010), seu principal mecanismo de busca nunca sofreu uma interrupção em todo esse tempo e seus usuários podem esperar resultados de consulta em torno de 0,2 segundos [googleblog.blogspot.com I]. O estudo de caso apresentado aqui examinará as estratégias e decisões de projeto por trás desse sucesso e dará uma ideia do projeto de sistemas distribuídos complexos.

O mecanismo de busca do Google

A função do mecanismo de busca do Google, assim como para qualquer mecanismo de pesquisa na Web, é receber determinada consulta e retornar uma lista ordenada com os resultados mais relevantes que satisfazem essa consulta, pesquisando o conteúdo da Web. Os desafios derivam do tamanho da Web e de sua taxa de mudança, assim como do requisito de fornecer os resultados mais relevantes do ponto de vista de seus usuários.

O mecanismo de busca consiste em um conjunto de serviços para esquadrinhar a Web e indexar e classificar as páginas descobertas [...].

Google como provedor de nuvem

O Google diversificou significativamente e, agora, além de pesquisa oferece uma ampla variedade de aplicações baseadas na Web, incluindo o conjunto promovido como Google Apps [www.google.com]. Mais geralmente, o Google agora é um forte participante na área da computação em nuvem. Lembre-se de que a computação em nuvem [...] “um conjunto de serviços de aplicativos, armazenamento e computação baseados na Internet, suficientes para suportar as necessidades da maioria dos usuários, permitindo, assim, que eles prescindam em grande medida ou totalmente do software local de armazenamento de dados ou de aplicativo”. É exatamente isso que agora o Google se esforça em oferecer, em particular com produtos significativos nas áreas de software como serviço e plataforma como serviço [...].

FONTE: COULOURIS, G. et al. **Sistemas distribuídos**: conceitos e projeto. 5. ed. Porto Alegre: Bookman, 2013. p. 917-955. Disponível em: <https://sdpisutic.files.wordpress.com/2017/08/sistemas-distribuc3addos-conceitos-e-projeto-2013.pdf>. Acesso em: 27 dez. 2018.

RESUMO DO TÓPICO 4

Neste tópico, você aprendeu que:

- Sistemas distribuídos baseados em objetos adotam o paradigma de orientação a objetos usado no desenvolvimento de sistemas.
- Sistemas distribuídos baseados em objetos tem sua arquitetura baseada em objetos distribuídos.
- Objetos distribuídos estabelecem a separação entre interfaces e os objetos que as implementam.
- Em sistemas distribuídos baseados em objetos a comunicação se dá através da vinculação entre um cliente e um objeto para posterior invocação de seus métodos. Esse processo recebe o nome de RMI.
- Sistemas de arquivos distribuídos seguem uma arquitetura cliente/servidor.
- Serviço de arquivo remoto permite aos clientes transparência de acesso aos servidores de arquivos remotos.
- Em sistemas de arquivos distribuídos a comunicação se dá através de RPC.
- Em sistemas de arquivos distribuídos a replicação do lado do servidor não costuma ser comum, diferentemente da replicação do lado do cliente, na qual o serviço de cache é largamente utilizado.
- No tocante à segurança, os sistemas de arquivos distribuídos devem realizar a autenticação das requisições dos clientes para controlar o acesso aos servidores.
- Tradicionalmente, sistemas baseados na web possuem arquitetura cliente/servidor, que evoluiu de duas camadas para três camadas.
- Serviços web fornecem uma infraestrutura mais rica de interoperabilidade entre clientes e servidores.
- HTTP é o protocolo utilizado na comunicação entre cliente e servidor para a apresentação de documentos web através do navegador, ao passo que SOAP é o protocolo para padronização da comunicação para serviços.
- O uso de *cache* e replicação de conteúdos auxiliam muito para a melhoria de desempenho dos sistemas distribuídos baseados na *web*.

- Sistemas distribuídos baseados na *web* são tolerantes a falhas à medida que implementam *cache* do lado do cliente e replicação dos servidores.
- Sistemas multimídia distribuídos são considerados sistemas de tempo real.
- Dados multimídia são caracterizados por serem contínuos e baseado no tempo.
- Qualidade de serviço é a característica de sistemas multimídia distribuídos de realizarem suas tarefas em tempo.
- Gerenciamento da qualidade de serviço envolve a capacidade de alocação de recursos, como largura de banda, por exemplo, já que as aplicações multimídia concorrem entre si.
- Gerenciamento de recursos ocorre através de métodos de escalonamento de recursos.



Ficou alguma dúvida? Construimos uma trilha de aprendizagem pensando em facilitar sua compreensão. Acesse o QR Code, que levará ao AVA, e veja as novidades que preparamos para seu estudo.





1 A construção do *middleware* de sistemas distribuídos é caracterizada pelo seu alto grau de abstração, o que permite a existência de vários paradigmas com arquiteturas próprias e diferentes formatos de comunicação. Isso pode ser verificado na existência de diferentes implementações de sistemas distribuídos, como os baseados em objetos e arquivos distribuídos, os baseados na web, e os sistemas de multimídia distribuídos. Com isso em mente, avalie as sentenças relacionadas à construção de sistemas distribuídos classificando com V as sentenças verdadeiras e F as falsas:

- () Apesar de os sistemas distribuídos baseados em objetos possuírem uma arquitetura baseada em objetos distribuídos esta nada tem a ver com o conceito de orientação a objetos, pois, apesar de os objetos encapsularem dados, seus métodos não são disponibilizados por meio de interfaces.
- () Em sistemas de arquivos distribuídos o modelo de acesso remoto permite ao cliente baixar e acessar os arquivos localmente. Após as atualizações, o cliente faz o upload do arquivo atualizado para o servidor.
- () A arquitetura cliente/servidor típica dos sistemas baseados na web evoluiu de duas para três camadas, na qual encontramos um servidor web, um servidor de aplicação e um servidor de banco de dados.
- () A comunicação entre sistemas distribuídos baseados na web é bastante simples, pois ocorre via HTTP apenas.

Agora, assinale a alternativa CORRETA:

- a) () V – V – F – V.
- b) () F – F – F – V.
- c) () F – F – V – F.
- d) () V – V – F – F.

2 A base dos sistemas distribuídos é o compartilhamento. A implementação de sistemas de arquivos distribuídos permite que usuários possam acessar e armazenar arquivos de dados a partir de qualquer dispositivo em uma rede exatamente como se o estivessem fazendo localmente, tendo-se ainda garantidos desempenho e controle de acesso. Sendo assim, avalie as sentenças relacionadas à construção de sistemas de arquivos distribuídos classificando com V as sentenças verdadeiras e F as falsas:

- () Sistemas de arquivos distribuídos seguem uma arquitetura cliente/servidor sendo o NFS a arquitetura mais utilizada.
- () Sistemas de arquivos distribuídos seguem uma arquitetura cliente/servidor sendo a RPC a arquitetura mais utilizada.

- () No tocante à consistência e replicação para sistemas de arquivos distribuídos, tanto a replicação do lado do servidor quanto os serviços de *cache* de dados no lado do cliente são estratégias comuns.
- () No tocante à consistência e replicação para sistemas de arquivos distribuídos, a replicação do lado do servidor não costuma ser uma estratégia comum, diferentemente dos serviços de *cache* de dados no lado do cliente.

Agora, assinale a alternativa CORRETA:

- a) () V – V – F – F.
- b) () F – V – V – F.
- c) () V – V – F – F.
- d) () V – F – F – V.

3 A construção do *middleware* de sistemas distribuídos é caracterizada pelo seu alto grau de abstração, o que permite a existência de vários paradigmas com arquiteturas próprias e diferentes formatos de comunicação. Isso pode ser verificado na existência de diferentes implementações de sistemas distribuídos, como os baseados em objetos e arquivos distribuídos, os baseados na web, e os sistemas de multimídia distribuídos. Com isso em mente, avalie as sentenças relacionadas à construção de sistemas distribuídos:

- I- Sistemas multimídia distribuídos devem ser capazes de manipular grandes volumes de dados, como vídeos em alta definição, por exemplo, sem perder o desempenho.
- II- Sistemas distribuídos baseados na web tem como principal característica o fato de competirem com outras aplicações por largura de banda.
- III- A comunicação em sistemas distribuídos baseados em objetos se dá através da vinculação de um cliente a um processo invocando os métodos do objeto via *proxy*. É o que chamamos de invocação de método remoto ou RMI.
- IV- Uma das formas de RMI é a invocação estática, na qual o método que será invocado é selecionado em tempo de execução.

Agora, indique a alternativa CORRETA:

- a) () As sentenças I e III estão corretas.
- b) () As sentenças II e IV estão corretas.
- c) () As sentenças I e II estão corretas.
- d) () As sentenças III e IV estão corretas.

4 A base dos sistemas distribuídos é o compartilhamento. A implementação de sistemas de arquivos distribuídos permite que usuários possam acessar e armazenar arquivos de dados a partir de qualquer dispositivo em uma rede exatamente como se o estivessem fazendo localmente, tendo-se ainda garantidos desempenho e controle de acesso. Sendo assim, avalie as sentenças relacionadas à construção de sistemas de arquivos distribuídos:

- I- Procedimentos compostos, caracterizados pelo agrupamento de várias RPC em uma única requisição consistem em uma técnica de replicação no lado do cliente.
- II- Procedimentos compostos, caracterizados pelo agrupamento de várias RPC em uma única requisição é uma forma de tornar sistemas de arquivos distribuídos tolerantes a falhas.
- III- O acesso a arquivos remotos se dá através de dois modelos: modelo de acesso remoto e modelo de carga/atualização.
- IV- O compartilhamento de arquivos distribuídos se dá pela implementação de um serviço de arquivo remoto, capaz de permitir a transparência de acesso às máquinas clientes.

Agora, indique a alternativa CORRETA:

- a) () As sentenças I e III estão corretas.
- b) () As sentenças II e IV estão corretas.
- c) () As sentenças I e II estão corretas.
- d) () As sentenças III e IV estão corretas.

5 Desde sua criação pelo CERN, a web passou de um sistema baseado em documentos estáticos para um grande sistema distribuído, acessado por milhões de usuários no mundo todo. Mais recentemente, com o compartilhamento dos serviços percebemos que a web passou a promover uma forma mais rica de interoperabilidade aos seus usuários. Sendo assim, avalie as sentenças relacionadas à construção de sistemas distribuídos baseados na web classificando com V as sentenças verdadeiras e F as falsas:

- () Para que um serviço web possa ser consumido, a comunicação entre os processos de uma máquina cliente e de uma máquina servidora ocorre via protocolo SOAP.
- () Sistemas distribuídos baseados na web possuem uma arquitetura cliente/servidor que se utiliza de um navegador para apresentar suas aplicações denominadas documentos web.
- () Serviços web fazem uso de scripts do lado do servidor que processam o documento antes de enviá-lo ao cliente.
- () Em sistemas distribuídos baseados na web, navegadores web e *proxies* web são os principais processos do lado do servidor.

Agora, assinale a alternativa CORRETA:

- a) () V – V – F – F.
- b) () F – F – V – V.
- c) () F – V – F – V.
- d) () V – F – F – F.

REFERÊNCIAS

- CIRILO, C. E. **Computação ubíqua**: definição, princípios e tecnologias. 2006. Disponível em: <https://docplayer.com.br/2317000-Computacao-ubiqua-definicao-principios-e-tecnologias.html>. Acesso em: 27 dez. 2018.
- COHEN, M. **Trabalhando com applets**: prática em laboratório. Porto Alegre: PUCRS, 2001. Disponível em: https://www.inf.pucrs.br/flash/lapro2/aula_applets.html. Acesso em: 5 jan. 2019.
- COULOURIS, G. et al. **Sistemas distribuídos**: conceitos e projeto. 5. ed. Porto Alegre: Bookman, 2013. Disponível em: <https://sdpisutic.files.wordpress.com/2017/08/sistemas-distribuc3addos-conceitos-e-projeto-2013.pdf>. Acesso em: 27 dez. 2018.
- DALLABONA-FARINIUK, T.; FIRMINO, R. Smartphones, smart spaces? o uso de mídias locativas no espaço urbano em Curitiba, Brasil. **EURE (Santiago)**, Santiago, v. 44, n. 133, p. 255-275, set. 2018. Disponível em: https://scielo.conicyt.cl/scielo.php?script=sci_arttext&pid=S0250-71612018000300255&lng=es&nrm=iso. Acesso em: 7 mar. 2019.
- DEITEL, H. M.; DEITEL, P. J.; CHOIFFNES, D. R. **Sistemas operacionais**. 3. ed. São Paulo: Pearson Prentice Hall, 2005.
- DEITEL, P.; DEITEL, H. **Java**: como programar. 10. ed. São Paulo: Pearson Education do Brasil, 2017. Disponível em: https://www.academia.edu/36118340/Java_-_Como_Programar_10a_Ed_Deitel_and_Deitel_2016_.pdf. Acesso em: 27 dez. 2018.
- INDEX MUNDI. **Número de servidores internet**: mundo. 2018a. Disponível em: <https://www.indexmundi.com/map/?t=0&v=140&r=xx&l=pt>. Acesso em: 5 set. 2019.
- JONES, S. C. Anatomy of a java applet: part 1. **Micro Focus**, Santa Clara, CA, ago, 1996. Disponível em: <https://support.novell.com/techcenter/articles/dnd19960801.html>. Acesso em: 5 jan. 2019.
- LAVADO, T. Uso da internet no Brasil cresce, e 70% da população está conectada. **G1**, São Paulo, 28 ago. 2019. Disponível em: <https://g1.globo.com/economia/tecnologia/noticia/2019/08/28/uso-da-internet-no-brasil-cresce-e-70percent-da-populacao-esta-conectada.ghtml>. Acesso em: 30 mar. 2020.

LEITHARDT, V. R. Q. *et al.* Uma proposta de classificação de dados para gerenciamento de privacidade em ambientes ubíquos utilizando o modelo UbiPri. In: CONGRESO INTERNACIONAL INFORMÁTICA EDUCATIVA, 19. 2014, Fortaleza, *Anais...* Fortaleza, TISE, 2014, p. 481-486. Disponível em: http://www.tise.cl/volumen10/TISE2014/tise2014_submission_46.pdf. Acesso em: 7 mar. 2019.

LEITHARDT, V. R. Q. *et al.* Uma proposta de classificação de dados para gerenciamento de privacidade em ambientes ubíquos utilizando o modelo UbiPri. In: CONGRESO INTERNACIONAL INFORMÁTICA EDUCATIVA, 19. 2014, Fortaleza, *Anais...* Fortaleza, TISE, 2014, p. 481-486. Disponível em: http://www.tise.cl/volumen10/TISE2014/tise2014_submission_46.pdf. Acesso em: 7 mar. 2019.

MACHADO, F. B. **Fundamentos de sistemas operacionais**. Rio de Janeiro: LTC, 2011.

MACHADO, F. B.; MAIA, L. P. **Arquitetura de sistemas operacionais**. 5. ed. Rio de Janeiro: LTC, 2017.

MACHADO, F. B.; MAIA, L. P. **Fundamentos de sistemas operacionais**. Rio de Janeiro: LTC, 2011.

O QUE é virtualização? **Redhat**, São Paulo, jun. 2018. Disponível em: <https://www.redhat.com/pt-br/topics/virtualization/what-is-virtualization>. Acesso em: 3 jan. 2019.

OLIVEIRA, R. S.; CARISSIMI, A. S.; TOSCANI, S. S. **Sistemas operacionais**. 4. ed. Porto Alegre: Bookman, 2010.

PATTERN. In: CAMBRIDGE Advanced Learner's Dictionary and Thesaurus. Cambridge: Cambridge University, 2018. Disponível em: <https://dictionary.cambridge.org/dictionary/english/pattern>. Acesso em: 17 dez. 2018.

PFLEEGER, C. P.; PFLEEGER, S. L. **Security in computing**. 3. ed. Upper Saddle River: Prentice Hall, 2003. Disponível em: https://books.google.com.br/books?id=O3VB-zspJo4C&printsec=frontcover&hl=pt-BR&source=gbs_ViewAPI&redir_esc=y#v=onepage&q=threat%20type&f=false. Acesso em: 24 fev. 2018.

SANTAELLA, L. **Comunicação ubíqua**: repercuções na cultura e na educação. São Paulo: Paulus, 2013. Disponível em: <https://books.google.com.br/books?id=h9y5DAAAQBAJ&pg=PT11&dq=computa%C3%A7%C3%A3o+ubiqua&hl=en&sa=X&ved=0ahUKEwif4Yuvy8DfAhVFGZAKHR4ZCjIQ6AEIKzAA#v=onepage&q=computa%C3%A7%C3%A3o%20ubiqua&f=false>. Acesso em: 27 dez. 2018.

SEO, C. E. **Virtualização**: problemas e desafios. Hortolândia: IBM Linux Technology Center, 2009. Disponível em: <https://www.ic.unicamp.br/~ducatte/mo401/1s2009/T2/008278-t2.pdf>. Acesso em: 7 mar. 2019.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Fundamentos de sistemas operacionais**. 9. ed. Rio de Janeiro: LTC, 2015. Disponível em: https://issuu.com/grupogen/docs/fundamentos_sistemas_operacionais. Acesso em: 28 out. 2018.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Sistemas operacionais com Java**. 7. ed. Rio de Janeiro: Elsevier, 2008. Disponível em: <https://books.google.com.br/books?id=Y8HMA2XngmwC&pg=PA37&dq=sistemas+operacionais+monotarefa&hl=en&sa=X&ved=0ahUKEwja-qyh8KneAhUBD5AKHcWyChYQ6AEIRjAE#v=onepage&q=sistemas%20operacionais%20monotarefa&f=false>. Acesso em: 28 out. 2018.

SILVA, E. et al. Computação ubíqua: definições e exemplos. **Rev. de Empreendedorismo, Inovação e Tecnologia**, Passo Fundo, v. 2, n. 1, p. 23-32, 2015. Disponível em: <https://seer.imed.edu.br/index.php/revistasi/article/view/926/739>. Acesso em: 7 mar. 2019.

STALLINGS, W. **Criptografia e segurança de redes**: princípios e práticas. 6. ed. São Paulo: Pearson Education do Brasil, 2015. (Pearson).

TANENBAUM, A. S.; STEEN, M. V. **Sistemas distribuídos**: princípios e paradigmas. 2. ed. São Paulo: Pearson Prentice Hall, 2007.

TANENBAUM, A. S.; WOODHULL, A. S. **Sistemas operacionais**: projeto e implementação. 3. ed. Porto Alegre: Bookman, 2008. Disponível em: <https://docero.com.br/doc/scx8cv>. Acesso em: 28 out. 2018.

THE JAVA EE 6 tutorial: what is messaging? **Oracle**, Redwood City, CA, 2010. Disponível em: <https://docs.oracle.com/cd/E19798-01/821-1841/bncds/index.html>. Acesso em: 27 dez. 2018.

TURBAN, E.; RAINER JR., R. K.; POTTER, R. E. **Administração de tecnologia da informação**: teoria e prática. Rio de Janeiro: Campus, 2003.

VIRTUALIZAÇÃO. VMware, Palo Alto, CA, 2019. Disponível em: <https://www.vmware.com/br/solutions/virtualization.html>. Acesso em: 3 jan. 2019.