Unidade IV

7 SISTEMA DE ARQUIVOS

O sistema de arquivos é responsável pelo fornecimento de mecanismo para armazenamento e acesso on-line tanto para os dados quanto para programas do SO e de todos os usuários do sistema de computação. Adicionalmente, trata-se do aspecto mais visível do SO para a maior parte dos usuários, pois existe uma interface de acesso a um arquivo que possui um conjunto de funções para manipular, como localizar o arquivo no dispositivo físico, realizar a leitura de seu conteúdo e verificar seus atributos, entre outras.

No sistema de arquivos, existem duas partes distintas: conjunto de arquivos e estrutura de diretórios, que organiza e fornece informações sobre todos os arquivos no sistema. De acordo com Silberschatz, Galvin e Gagne (2015), todo e qualquer arquivo contém um conjunto de dados associados que foram registrados em sua memória secundária. Cada arquivo possui seus dados relacionados definidos por seus criadores.

Sistemas de arquivos é o nome atribuído à organização do conteúdo dos arquivos e diretórios em um dispositivo físico. Existe uma variedade de sistemas de arquivos, e entre eles temos: o NTFS para Windows, Ext2/ Ext3/Ext4 para Linux, HPFS para Mac OS, FFS para Solaris e FAT utilizado em pendrives, USB, câmeras fotográficas digitais e leitores MP3.

7.1 Atributos dos arquivos

Visando à conveniência dos usuários, um arquivo é nomeado e referenciado por esse nome. Um nome é, usualmente, uma sequência de caracteres, como em exemplo.c. Existem sistemas de arquivos que diferenciam caracteres maiúsculos e minúsculos nos nomes, isto é, são case sensitive, e outros não têm essa diferenciação.

Após a nomeação do arquivo, ele se torna independente do processo, do usuário e até mesmo do sistema que o criou. Por exemplo, um usuário poderia criar o arquivo exemplo.c, e outro editá-lo especificando seu nome. O proprietário do arquivo poderia gravá-lo em um disco USB, enviá-lo como um anexo de e-mail ou copiá-lo por meio de uma rede, e, mesmo assim, ele continuaria sendo chamado exemplo.c no sistema de destino.

Existem variações entre os atributos de um arquivo entre SOs, mas atributos típicos são o nome, o identificador, o tipo, a locação, o tamanho, a proteção e a hora, a data de criação e a identificação do usuário.

O nome simbólico do arquivo é a única informação mantida de forma legível por humanos e cada sistema de arquivos possui regras sobre a quantidade máxima de caracteres e quais deles são aceitos. O identificador possui um rótulo exclusivo, usualmente um número, identifica o arquivo no sistema de arquivos e não é legível por humanos.

Outro atributo relevante é o tipo do arquivo. Essa informação é necessária para sistemas que suportam diferentes tipos de arquivos. A locação representa um ponteiro para um dispositivo e para a locação do arquivo nesse dispositivo. Cada arquivo possui um tamanho, que pode ser expresso em bytes, palavras ou blocos e o SO define um tamanho máximo permitido. Por exemplo, o sistema de arquivos informou que o arquivo relatorio.txt possui 11.349 bytes.

No atributo de proteção, são disponibilizadas as informações de controle de acesso que determinam se o usuário pode ler, gravar, executar o arquivo, e assim por diante. Outros atributos relevantes são a hora, a data e a identificação do usuário que criou o arquivo, que o alterou pela última vez, bem como o último. Esses dados podem ser utilizados para proteção, segurança e monitoramento do uso.

7.2 Tipos de arquivos

As informações em um arquivo são definidas por seu criador. Muitos tipos diferentes de informações podem ser armazenados em um arquivo — programas-fonte ou executáveis, dados numéricos ou de texto, fotos, música, vídeo, e assim por diante. O arquivo tem uma estrutura específica definida, que depende de seu tipo. Um arquivo de texto é uma sequência de caracteres organizada em linhas (e possivelmente páginas). Um arquivo-fonte é uma sequência de funções, todas organizadas como declarações seguidas por comandos executáveis. Um arquivo executável é uma série de seções de código que o carregador pode trazer para a memória e executar.

Os sistemas de arquivos residem em dispositivos que já descrevemos e continuaremos a fazê-lo posteriormente. Neste tópico consideraremos os diversos aspectos dos arquivos e as principais estruturas de diretórios. Também abordaremos a semântica de compartilhamento de arquivos entre múltiplos processos, usuários e computadores. Finalmente, lidaremos com formas de manipular a proteção de arquivos, necessária quando temos múltiplos usuários e queremos controlar quem pode acessar arquivos e como eles podem ser acessados.

Ao projetar um sistema de arquivos, sempre consideramos se o SO deve reconhecer e suportar quais tipos de arquivo. Um SO somente pode operar sobre o arquivo se reconhecer o tipo desse arquivo. Por exemplo, um erro comum ocorre quando um usuário tenta dar saída na forma objeto binária de um programa. Normalmente essa tentativa produz lixo; no entanto, a tentativa pode ser bem-sucedida se o SO tiver sido informado de que o arquivo é um programa-objeto binário.

Uma técnica comum para a implementação dos tipos de arquivo é incluir o tipo como parte do nome do arquivo. O nome é dividido em duas partes — um nome e uma extensão, usualmente separados por um ponto. Dessa forma, o usuário e o SO podem identificar, somente pelo nome, qual é o tipo de um arquivo. A maioria dos SOs permite que os usuários especifiquem um nome de arquivo como uma

sequência de caracteres seguida por um ponto e encerrada por uma extensão composta de caracteres adicionais. Exemplos incluem resumo.docx, servidor.c e ReaderThread.cpp.

O sistema utiliza a extensão para indicar o tipo do arquivo e o tipo de operações que podem ser realizadas com ele. Somente um arquivo com uma extensão .com, .exe ou .sh pode ser executado, por exemplo. Os arquivos .com e .exe são binários executáveis, enquanto o .sh é de um script de shell contendo comandos para o SO em formato ASCII. Programas de aplicação também usam extensões para indicar os tipos de arquivos nos quais estão interessados.

No Linux, um arquivo binário possui um conteúdo que apenas pode ser entendido computacionalmente, pois contém uma cadeia de caracteres incompreensíveis para os usuários. Esses arquivos são gerados com base em um programa-fonte através de um processo chamado de compilação. Conforme visto, a compilação pode ser entendida como a conversão de um programa em linguagem de programação para a linguagem de máquina.

Por exemplo, os compiladores Java esperam que os arquivos-fonte tenham uma extensão .java, e o processador de textos Microsoft Word que seus arquivos terminem com uma extensão .doc ou .docx. Nem sempre essas extensões são exigidas; portanto, um usuário pode especificar um arquivo elas (para economizar digitação), e a aplicação procurará por um arquivo com o nome dado e a extensão esperada. Como essas extensões não são suportadas pelo SO, elas podem ser consideradas como "dicas" para as aplicações que as manipulam.

Um dos tipos de arquivos mais frequentemente utilizados é o de texto puro ou plain text. Ele é usado para armazenamento de informações textuais simples, tais como códigos-fonte de programas, arquivos de configuração, registros de log, páginas HTML, dados em XML, entre outros. Um arquivo de texto é composto de linhas de caracteres de tamanho variável, separadas por caracteres de controle.

Observação

Em Unix, as linhas de arquivos textos são separadas por um caractere New Line, cujo código ASCII é 10 e é representado por "\n". Para DOS/Windows, a separação das linhas de um arquivo de texto é feita por dois caracteres: o caractere Carriage Return, cujo código ASCII é 13 e representado por "\r", seguido de New Line. O código ASCII tem a função de padronizar a forma como os computadores representam letras, números, acentos, sinais diversos e alguns códigos de controle.

O quadro a seguir apresenta alguns tipos de arquivos e suas extensões utilizadas.

Quadro 4 – Principais tipos de arquivos

Tipo de arquivo	Extensão	Função
Arquivo executável	.exe, .com, .bin	Programa em linguagem de máquina pronto para ser executado
Arquivos de texto	.txt	Arquivos de texto sem formatação
Arquivo no formato PDF (Portable Document Format)	.pdf	Representação de documentos de maneira independente do aplicativo, hardware e SO usados para criá-los
Arquivamento	.rar, .zip, .tar	Arquivos relacionados compactados em um único arquivo
Processador de texto	.xml, .rtf, .docx	Vários formatos de processadores de texto
Multimídia	.mpeg, .mov, mp3, .mp4	Arquivos binários com informação de áudio e vídeo
Fotografia	.jpg, .png, .bmp	Fotografia codificada em diferentes padrões de imagem
Código-fonte	.c, .java, .perl	Programa-fonte em várias linguagens de programação

Adaptado de: Silberschatz, Galvin e Gagne (2015, p. 273).

O Unix usa um número mágico simples, magic number, armazenado no começo de alguns arquivos para indicar grosseiramente o tipo do arquivo – programa executável, script de shell, PDF, e assim por diante. Nem todos os arquivos têm números mágicos. Consequentemente, portanto, os recursos do sistema não podem se basear apenas nessa informação. O Unix também não registra o nome do programa criador. Ele permite o uso de dicas para extensões de nomes de arquivos, mas não as impõe nem depende delas; sua principal finalidade é ajudar os usuários a determinar o tipo de conteúdo que o arquivo contém. As extensões podem ser usadas ou ignoradas por determinada aplicação, mas quem decide isso é o programador da aplicação. A tabela a seguir apresenta os tipos de arquivos e os números mágicos.

Tabela 4 – Números mágicos de alguns tipos de arquivo

Tipo de arquivo	Bytes iniciais
Imagem PGM ascii	P2\n
Documento PDF	%PDF
Imagem GIF	GIF89a
Imagem JPEG	0xFF D8 FF
Música MIDI	MThd
Classes Java	0xCA FE BA BE
Arquivo ZIP	0x50 4B 03 04
Documento RTF	{\rtf1

Fonte: Maziero (2019, p. 285).

7.3 Operações de E/S

O sistema de arquivos disponibiliza um conjunto de rotinas que permitem às aplicações realizarem operações de E/S, como tradução de nomes em endereços, leitura e gravação de dados, criação e eliminação de arquivos. O acesso aos arquivos é efetuado por meio de um conjunto de operações implementadas através de chamadas de sistemas e funções de bibliotecas.

As rotinas de E/S têm como função disponibilizar uma interface simples e uniforme entre a aplicação e os diversos dispositivos.

O SO pode fornecer chamadas de sistema para criar, gravar, ler, reposicionar, excluir e truncar arquivos. Examinaremos o que ele precisa fazer para executar cada uma dessas seis operações básicas de arquivo. Assim, ficará fácil ver como outras operações semelhantes, como a renomeação de um arquivo, podem ser implementadas.

Para a criação de um arquivo, são necessários dois passos: encontrar espaço para o arquivo no sistema de arquivos e criar uma entrada para o novo arquivo no diretório.

Na gravação de um arquivo, é feita uma chamada de sistema especificando tanto o nome do arquivo quanto as informações a serem nele gravadas. Dado o nome do arquivo, o sistema pesquisa o diretório para determinar a locação do arquivo. O sistema deve manter um ponteiro de gravação para a locação no arquivo em que a próxima gravação tem de ocorrer. O ponteiro de gravação precisa ser atualizado sempre que ocorrer uma gravação.

Na leitura de um arquivo, é utilizada uma chamada de sistema que especifica o nome do arquivo e onde o próximo bloco do arquivo deve ser salvo na memória. Novamente, o diretório é pesquisado em busca da entrada associada, e o sistema precisa manter um ponteiro de leitura para a locação no arquivo em que a próxima leitura deve ocorrer. Uma vez que a leitura tenha ocorrido, o ponteiro de leitura é atualizado. Já que um processo usualmente tanto faz leituras quanto gravações em um arquivo, a locação da operação corrente pode ser mantida como um ponteiro da posição corrente do arquivo por processo. Tanto as operações de leitura quanto as de gravação usam o mesmo ponteiro, economizando espaço e reduzindo a complexidade do sistema.

Outra operação é o reposicionamento dentro de um arquivo, na qual o diretório é pesquisado em busca da entrada apropriada, e o ponteiro da posição corrente é reposicionado para determinado valor. Ele não precisa envolver nenhuma E/S real. Essa operação de arquivo também é denominada busca em arquivo.

Para ocorrer a exclusão de um arquivo, é necessário pesquisar o diretório em busca do arquivo nomeado. Após encontrar a entrada associada no diretório, libera-se todo o espaço do arquivo, para que ele possa ser reutilizado por outros arquivos, e apaga-se a entrada no diretório.

Considerando a operação de truncamento de um arquivo, o usuário pode querer apagar o conteúdo de um arquivo, mas manter seus atributos, como a data/hora de criação. Com isso, ele não precisará excluir o arquivo e depois recriá-lo, pois essa função permite que todos os atributos permaneçam inalterados, com exceção ao tamanho, uma vez que o resultado será o arquivo redefinido com o tamanho zero e seu espaço anteriormente ocupado liberado.

Além dessas operações básicas de arquivos, há operações comuns que incluem o acréscimo de novas informações ao fim de um arquivo existente e a renomeação de um arquivo existente. Essas operações primitivas podem então ser combinadas para executar outras operações de arquivo. Por exemplo, é

possível gerar uma cópia de um arquivo em outro dispositivo de I/O, como uma impressora ou um vídeo, ou criar um novo arquivo e, em seguida, lendo a partir do antigo e armazenando no novo.

Outra possibilidade é ter operações que permitam ao usuário obter e definir os diversos atributos de um arquivo. Por exemplo, podemos querer contar com operações que permitam a um usuário determinar o status de um arquivo, tal como seu tamanho, e definir atributos do arquivo, tal como seu proprietário.

A maior parte das operações de arquivo envolve uma pesquisa no diretório em busca da entrada associada ao arquivo nomeado. Para minimizar a quantidade dessas buscas, muitos sistemas requerem que uma chamada de sistema open ( ) seja realizada previamente à utilização de um arquivo pela primeira vez.

O SO alimentará uma tabela, denominada tabela de arquivos abertos, contendo as informações sobre todos os arquivos abertos naquele momento. Quando uma operação de arquivo é solicitada, ele estará determinado através de um índice presente nessa tabela. Assim, nenhuma busca adicional, que empregaria recursos desnecessários, será exigida. Quando o arquivo não for mais ativamente usado, este será fechado pelo processo, e o SO removerá sua entrada da tabela de arquivos abertos. As chamadas de sistema create ( ) e delete ( ) operam com arquivos fechados em vez de abertos.

Alguns sistemas abrem implicitamente um arquivo quando é feita a primeira referência a ele e o arquivo é fechado automaticamente quando o job ou o programa que o abriu termina. A maioria dos sistemas, no entanto, requer que o programador abra um arquivo explicitamente com a chamada de sistema open ( ) antes que esse arquivo possa ser usado. A operação open ( ) recebe um nome de arquivo e pesquisa o diretório, copiando a entrada do diretório na tabela de arquivos abertos.

Adicionalmente, a chamada open ( ) aceita informações sobre a modalidade de acesso, que pode ser de criação, somente leitura, leitura e gravação, somente de acréscimo, que é verificada em relação às permissões do arquivo. Se a modalidade solicitada for permitida, o arquivo será aberto para o processo. A chamada de sistema open ( ) retorna tipicamente um ponteiro para a entrada na tabela de arquivos abertos.

A implementação das operações open ( ) e close ( ) é mais complexa em um ambiente computacional onde vários processos podem abrir o arquivo simultaneamente. Isso pode ocorrer em um sistema em que várias aplicações diferentes abram o mesmo arquivo ao mesmo tempo. Normalmente, o SO usa dois níveis de tabelas internas: uma tabela por processo e outra para todo o sistema. A tabela por processo controla todos os arquivos que o processo abriu. Ela armazena informações relacionadas com o uso do arquivo pelo processo. Por exemplo, o ponteiro corrente para cada arquivo é encontrado aqui. Direitos de acesso ao arquivo e informações de contabilidade também podem ser incluídos.

Por sua vez, cada entrada na tabela por processo aponta para uma tabela de arquivos abertos em todo o sistema. A tabela para todo o sistema contém informações independentes do processo, tais como locação do arquivo em disco, datas de acesso e tamanho do arquivo. Uma vez que um arquivo tenha sido aberto por um processo, a tabela para todo o sistema incluirá uma entrada para o arquivo. Quando outro processo executar uma chamada open ( ), uma nova entrada será simplesmente adicionada à tabela

de arquivos abertos do processo apontando para a entrada apropriada na tabela para todo o sistema. Normalmente, a tabela de arquivos abertos também tem uma contagem de aberturas associada a cada arquivo para indicar quantos processos o abriram. Cada close ( ) diminui essa contagem de aberturas, e, quando a contagem alcança zero, o arquivo não está mais em uso, e sua entrada é removida da tabela de arquivos abertos.

Existem diversas informações associadas a um arquivo aberto. O sistema controla a última locação de leitura/gravação com um ponteiro do arquivo que indica a posição corrente do arquivo. Tal ponteiro é exclusivo para cada processo que operar sobre o arquivo e, portanto, deve ser mantido separado dos atributos do arquivo em disco.

O sistema também realiza a contagem de arquivos abertos e, conforme os arquivos são fechados, o SO deve reutilizar suas entradas na tabela de arquivos abertos ou pode ficar sem espaço na tabela. Vários processos podem ter aberto um arquivo, e o sistema deve esperar que o último arquivo seja fechado antes de remover a entrada na tabela de arquivos abertos. A contagem de arquivos abertos controla o número de aberturas e fechamentos e chega a zero no último fechamento.

As informações necessárias à localização do arquivo em disco são mantidas na memória para que o sistema não tenha que lê-las a partir do disco a cada operação, enquanto a maioria das operações de arquivo requer que o sistema modifique dados no arquivo.

Quanto aos direitos de acesso, cada processo abre um arquivo em uma modalidade de acesso e essa informação é armazenada na tabela por processo. Com isso, o SO consegue determinar se deve permitir ou negar solicitações de E/S subsequentes.

Alguns SOs fornecem recursos para trancamento de um arquivo aberto (ou de seções de um arquivo). Os locks de arquivo permitem que um processo tranque um arquivo impedindo que outros processos obtenham acesso a ele. Esses locks são úteis para arquivos que sejam compartilhados por vários processos — por exemplo, um arquivo de log do sistema que possa ser acessado e modificado por vários processos no sistema.

Os locks de arquivo fornecem funcionalidade semelhante à dos locks de leitor-gravador. Um lock compartilhado é parecido com um lock de leitor, já que vários processos podem adquirir o lock concorrentemente. Um lock exclusivo comporta-se como um lock de gravador; apenas um processo de cada vez pode adquirir esse tipo de lock. É importante observar que nem todos os SOs fornecem os dois tipos de locks; alguns sistemas fornecem somente o lock de arquivos exclusivo.

Os SOs também podem fornecer mecanismos de trancamento de arquivos obrigatórios ou aconselháveis. Se um lock é obrigatório, uma vez que um processo adquire um lock exclusivo, o SO impedirá que outro processo acesse o arquivo trancado. Por exemplo, suponha que um processo adquira um lock exclusivo para o arquivo system.log. Se tentarmos abrir system.log a partir de outro processo — por exemplo, um editor de texto — o SO impedirá o acesso até que o lock exclusivo seja liberado. Isso ocorre mesmo que o editor de texto não tenha sido escrito explicitamente para adquirir o lock. Alternativamente, se o lock é aconselhável, o SO não impedirá que o editor de texto obtenha acesso a

system.log. Em vez disso, o editor deve ser escrito para adquirir manualmente o lock antes de acessar o arquivo. Em outras palavras, se o esquema de trancamento é obrigatório, o SO assegura a integridade do trancamento.

A interface de baixo nível é formada por chamadas de sistema, posto que os arquivos são abstrações criadas e mantidas pelo kernel do SO. Assim, essa interface depende do SO subjacente. O quadro a seguir apresenta algumas chamadas de sistema oferecidas por Linux e Windows para o acesso a arquivos:

Quadro 5 – Chamadas de sistemas de arquivos

Operação	Linux	Windows
Abrir arquivo	OPEN	NtOpenFile
Ler dados	READ	NtReadRequestData
Escrever dados	WRITE	NtWriteRequestData
Fechar arquivo	CLOSE	NtClose
Remover arquivo	UNLIKE	NtDeleteFile
Criar diretório	MKDIR	NtCreateDirectoryObject

Fonte: Maziero (2019, p. 289).

Partições

As partições ou volumes representam divisões existentes no disco rígido que marcam onde começa e termina um sistema de arquivos. Através delas, é possível utilizar mais de um SO no mesmo computador, por exemplo, o GNU/Linux, Windows e DOS, ou dividir o disco rígido em uma ou mais partes para ser usado por um mesmo SO ou até mesmo por diferentes arquiteturas de 32 bits ou 64 bits.

Como cada volume pode armazenar um sistema de arquivos próprio, um mesmo disco pode conter volumes com diferentes sistemas de arquivos, como FAT, NTFS ou EXT4, por exemplo.

7.4 Arquitetura dos sistemas de arquivos

Os principais elementos que realizam a implementação de arquivos no SO estão organizados nas seguintes camadas: dispositivos físicos, controladores, drivers, gerência de blocos, alocação de arquivos, sistemas de arquivos virtuais, interface do sistema de arquivos e biblioteca de E/S. A figura a seguir apresenta a arquitetura geral do sistema de arquivos.

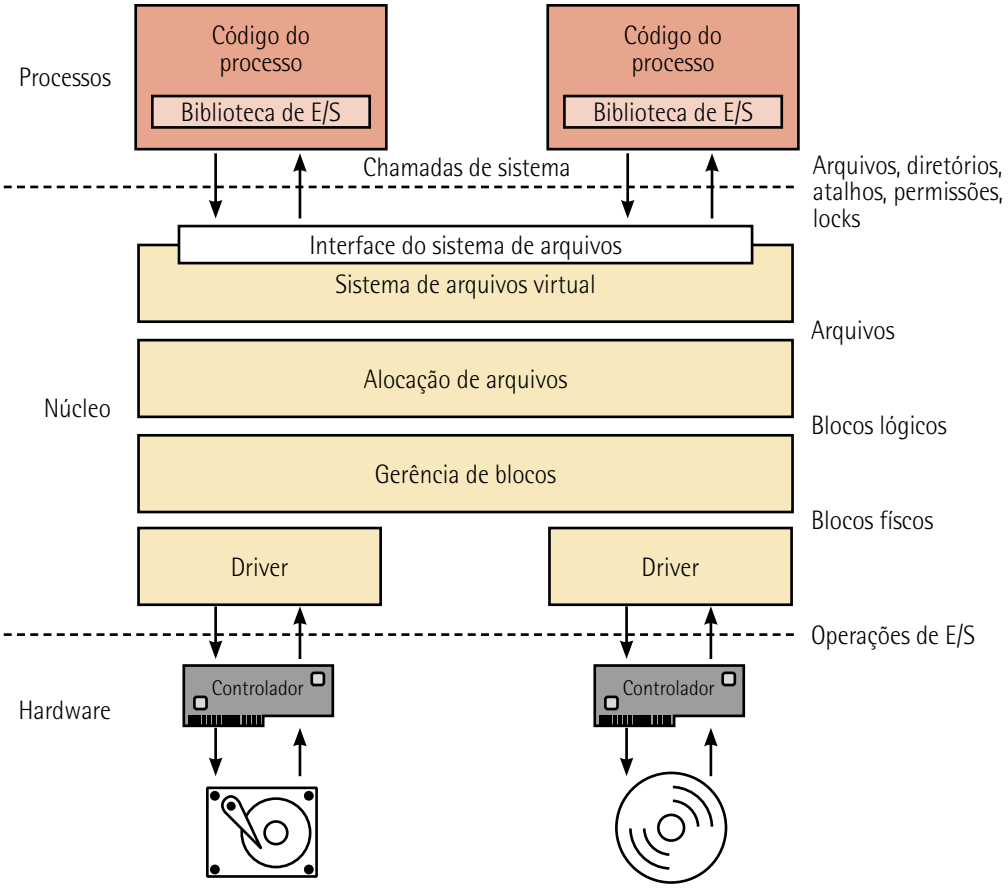

Figura 46 – Camada de implementação da gerência de arquivo

Fonte: Maziero (2019, p. 303).

Os dispositivos são representados por discos rígidos ou bancos de memória flash, sendo responsáveis pelo armazenamento de dados. Na camada de controladores, estão os circuitos eletrônicos dedicados ao controle dos dispositivos físicos. Para acessar os controladores, podem ser utilizadas portas de E/S, interrupções e canais de acesso direto à memória (DMA).

Os drivers são responsáveis pela interação entre os controladores de dispositivos, para configurá-los e realizar as transferências de dados entre o SO e os dispositivos. Como cada controlador define sua própria interface, também possui um driver específico. Eles ocultam as diferenças entre controladores e fornecem às camadas superiores do núcleo uma interface padronizada para acesso aos dispositivos de armazenamento.

Na gerência de blocos, ocorre a gerência do fluxo de blocos de dados entre as camadas superiores e os dispositivos de armazenamento. Como os discos são dispositivos orientados a blocos, as operações de leitura e escrita de dados são sempre feitas com blocos de dados, e nunca com bytes individuais. As funções mais importantes dessa camada são efetuar o mapeamento de blocos lógicos nos blocos físicos do dispositivo e o caching/buffering de blocos.

Por meio da alocação de arquivos, estes são colocados sobre os blocos lógicos oferecidos pela camada de gerência de blocos. Cada arquivo é visto como uma sequência de blocos lógicos que deve ser armazenada nos blocos dos dispositivos de forma eficiente, robusta e flexível.

Na camada do sistema de arquivos virtual ou virtual file system (VFS), são construídas as abstrações de diretórios e atalhos, além de gerenciar as permissões associadas aos arquivos e as travas de acesso compartilhado. Outra responsabilidade importante dela é manter o registro de cada arquivo aberto pelos processos, como a posição da última operação no arquivo, o modo de abertura usado e o número de processos que estão usando o arquivo.

A interface do sistema de arquivos é o conjunto de chamadas de sistema oferecidas aos processos do espaço de usuários para a criação e manipulação de arquivos. Por fim, bibliotecas de E/S utilizam as chamadas de sistema da interface do núcleo para construir funções padronizadas de acesso a arquivos para cada linguagem de programação.

7.4.1 Diretórios

Um diretório, pasta ou folder representa um repositório de arquivos e de outros diretórios. Da mesma forma que os arquivos, eles possuem nome e atributos que são usados na localização e acesso ao seu conteúdo. Em cada sistema de arquivos, há um diretório principal, denominado diretório raiz ou root directory.

Nos primórdios dos SOs, havia apenas um diretório, o diretório raiz, que continha todos os arquivos. Posteriormente foram implementados subdiretórios, ou seja, um nível de diretórios abaixo do diretório raiz. Os sistemas de arquivos atuais oferecem uma estrutura muito mais flexível, com um número de níveis de diretórios muito mais elevado, ou mesmo ilimitado (como nos sistemas de arquivos NTFS e Ext4).

Ao considerar uma estrutura de diretório específica, precisamos manter em mente as operações que são executadas em um diretório, como: busca de um arquivo, criação de um arquivo, exclusão de um arquivo, listagem do diretório, renomeação de um arquivo e varredura no sistema de arquivos.

Para a busca de um arquivo, precisamos ser capazes de pesquisar uma estrutura de diretório a fim de encontrar a entrada relacionada com um arquivo específico. Já que os arquivos têm nomes simbólicos, e nomes semelhantes podem indicar um relacionamento entre arquivos, podemos querer encontrar todos os arquivos cujos nomes correspondam a um padrão específico.

- **Criação de um arquivo**: novos arquivos precisam ser criados e adicionados ao diretório.
- **Exclusão de um arquivo**: quando um arquivo não é mais necessário, queremos ser capazes de removê-lo do diretório.
- **Listagem de um diretório**: precisamos ser capazes de listar os arquivos em um diretório e o conteúdo da entrada relacionada a ele com cada arquivo na lista.

- **Renomeação de um arquivo**: como o nome de um arquivo representa seu conteúdo para seus usuários, devemos ser capazes de alterar o nome quando o conteúdo ou o uso do arquivo mudar. A renomeação de um arquivo também pode permitir que sua posição na estrutura do diretório seja alterada.

- **Varredura no sistema de arquivos**: podemos querer acessar cada diretório e arquivo dentro de uma estrutura de diretório. A título de confiabilidade, é uma boa ideia salvar o conteúdo e a estrutura do sistema de arquivos inteiro em intervalos regulares. Geralmente, fazemos isso copiando todos os arquivos em fita magnética. Essa técnica fornece uma cópia de backup para o caso de falha do sistema. Além disso, se um arquivo não está mais em uso, ele pode ser copiado em fita, e o seu espaço em disco pode ser liberado e reutilizado por outro arquivo.

7.5 Estrutura lógica de um diretório

Existem diferentes formas de organizar a estrutura de diretórios para a definição da estrutura lógica de um diretório, como o diretório de um nível, diretórios de dois níveis, diretório estruturado em árvores e diretórios em grafos.

7.5.1 Diretório de um nível

A estrutura de diretório mais simples é o diretório de somente um nível, no qual todos os arquivos são colocados no mesmo diretório. Uma vantagem dessa estrutura é a facilidade de entendimento e de suporte do SO.

Entretanto, o diretório de um nível apresenta severas limitações conforme o aumento do número de arquivos ou quando da utilização do sistema por mais de um usuário. Como todos os arquivos estão no mesmo diretório, eles devem ter nomes exclusivos.

Logo, se dois usuários atribuírem o nome de trabalho.txt ao arquivo de dados e o colocarem no mesmo diretório, então a regra do nome exclusivo será violada. Por exemplo, em uma turma de programação, 21 estudantes chamaram de prog2.c o programa de seu segundo exercício; outros 13 o chamaram de exercicio2.c. Felizmente, a maioria dos sistemas de arquivos suporta nomes de arquivos com até 255 caracteres; assim, é relativamente fácil selecionar nomes de arquivos exclusivos. Sistemas mais antigos como o Microsoft DOS limitavam o nome do arquivo em somente oito caracteres para o nome e três caracteres para sua extensão, que representa o tipo de arquivo.

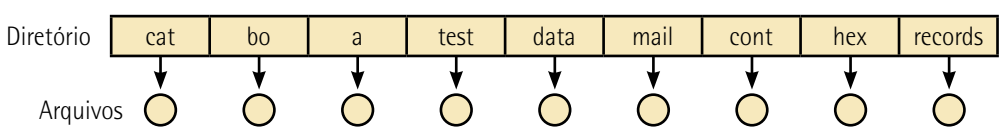

Figura 47 – Diretório de um nível

Adaptada de: Silberschatz, Galvin e Gagne (2015, p. 273).

Mesmo em situações de um único usuário em um diretório de um nível, pode ser difícil lembrar os nomes de todos os arquivos à medida que o número deles cresce. A maioria dos usuários possui centenas de arquivos em um sistema de computação e ainda existem muitos arquivos do próprio SO. Para um usuário comum, o controle de tantos arquivos é uma tarefa assustadora.

7.5.2 Diretório de dois níveis

Dado que diretório de um nível normalmente gera o conflito de nomes de arquivo entre diferentes usuários, uma solução frequente é a criação de um diretório separado para cada usuário.

Na estrutura de diretório de dois níveis, cada usuário possui seu próprio diretório de arquivos do usuário ou user file directory (UFD). Os UFDs têm estruturas semelhantes, mas cada um deles lista somente os arquivos de um usuário. Quando um job de usuário é iniciado ou um usuário faz login, o diretório de arquivos mestres ou master file directory (MFD) do sistema é pesquisado. O MFD é indexado por nome de usuário ou número de conta, e cada entrada aponta para o UFD dele, conforme apresentado na figura a seguir.

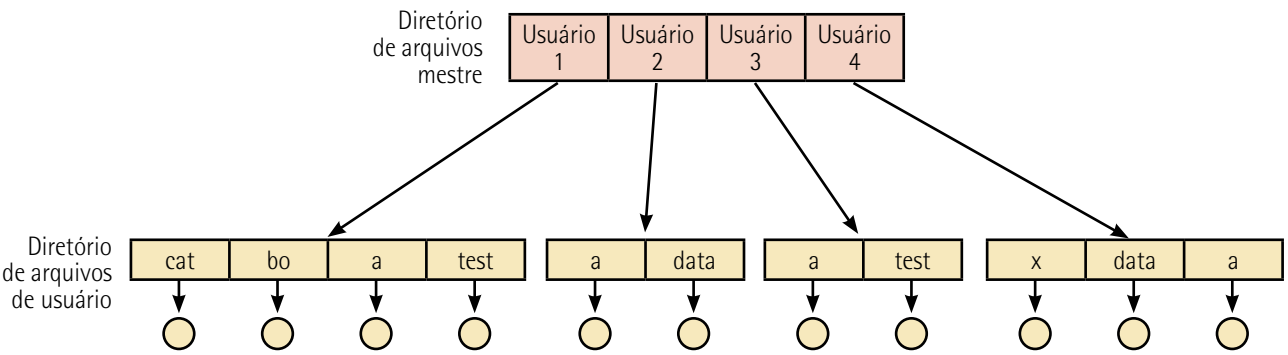

Figura 48 – Diretório de dois níveis

Adaptada de: Silberschatz, Galvin e Gagne (2015, p. 273).

Quando um usuário realiza uma pesquisa sobre um arquivo específico, somente seu próprio UFD é consultado. Consequentemente, diferentes usuários podem ter arquivos com o mesmo nome com a ressalva de que todos os nomes de arquivos, dentro de cada UFD, sejam únicos e exclusivos. No momento da criação de um arquivo para um usuário, o SO pesquisa exclusivamente o diretório de arquivos desse usuário para verificar a existência de outro arquivo com esse nome. Durante a exclusão de um arquivo, o SO realiza a pesquisa limitada ao UFD local. Dessa forma, um usuário não pode excluir acidentalmente o arquivo de outro ainda que tenha o mesmo nome.

Os próprios diretórios de usuário também devem ser criados e excluídos quando necessário. Um programa de sistema especial é executado com as informações apropriadas de nome e conta de usuário. O programa cria um novo UFD e adiciona uma entrada para ele no diretório de arquivos mestres (MFD). A execução desse programa pode estar restrita aos administradores do sistema.

A estrutura do diretório de dois níveis resolve o problema da colisão de nomes, entretanto esta estrutura ainda apresenta desvantagens significativas. Nela, há isolamento efetivo entre os diferentes usuários. Tal isolamento é vantajoso quando os usuários são totalmente independentes, mas é uma desvantagem quando eles precisam cooperar em alguma tarefa e acessar arquivos uns dos outros. Alguns sistemas simplesmente não permitem que arquivos locais de usuários sejam acessados por outros usuários.

Se o acesso for permitido, o usuário deve possuir a capacidade de nomear um arquivo no diretório de outro. Para nomear um arquivo específico de maneira exclusiva em um diretório de dois níveis, devemos fornecer tanto o nome do usuário quanto o nome do arquivo. Um diretório de dois níveis pode ser considerado como uma árvore, ou uma árvore invertida, de altura 2. A raiz da árvore é o MFD. Seus descendentes diretos são os UFDs. Os descendentes dos UFDs são os arquivos propriamente ditos. Os arquivos são as folhas da árvore. A especificação de um nome de usuário e um nome de arquivo define um caminho na árvore que vai da raiz (o MFD) até uma folha (o arquivo especificado). Portanto, um nome de usuário e um nome de arquivo definem um nome de caminho. Todo arquivo no sistema tem um nome de caminho. Para nomear um arquivo de maneira exclusiva, o usuário deve saber o nome de caminho do arquivo desejado.

Assim, se o usuário X quer acessar seu arquivo de teste chamado teste.txt, ele pode simplesmente referenciar teste.txt. Para acessar o arquivo chamado teste.txt do usuário B (com o nome da entrada no diretório userb), no entanto, ele pode ter que referenciar /userb/teste.txt. Cada sistema tem sua própria sintaxe para a nomeação de arquivos nos diretórios que é diferente da sintaxe do usuário.

7.5.3 Diretórios estruturados em árvore

O uso de diretórios permite construir uma estrutura hierárquica (em árvore) de armazenamento dentro de um volume e sobre a qual os arquivos são organizados.

Partindo-se da visualização de um diretório de dois níveis como uma árvore de dois níveis, a generalização natural é estender a estrutura de diretórios para uma árvore de altura arbitrária. Essa generalização permite que os usuários criem seus próprios subdiretórios e organizem seus arquivos de acordo. Uma árvore é a estrutura de diretório mais comum. Ela tem um diretório raiz, e cada arquivo no sistema tem um nome de caminho exclusivo.

A figura a seguir apresenta a estrutura de diretórios em árvore.

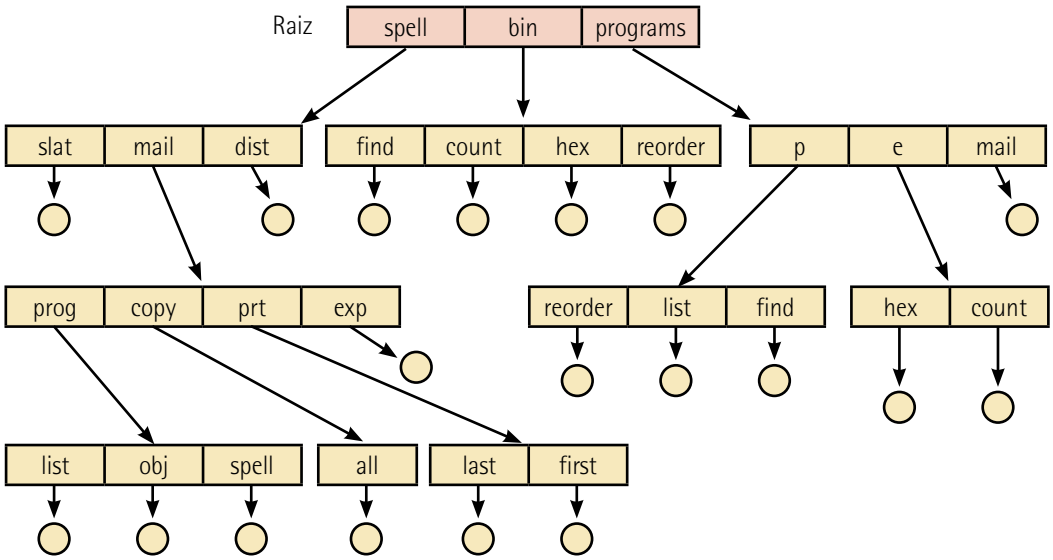

Figura 49 – Estrutura de diretório em árvore

Adaptada de: Silberschatz, Galvin e Gagne (2015, p. 273).

Um diretório ou subdiretório contempla um conjunto de arquivos ou subdiretórios. Trata-se simplesmente outro arquivo, porém com um tratamento especial. Todos os diretórios possuem o mesmo formato interno e um bit em cada entrada no diretório define a entrada como um arquivo (0) ou um subdiretório (1). Em cada SO, existem chamadas de sistema especiais usadas para criar e excluir diretórios.

Normalmente, cada processo tem um diretório corrente, e esse diretório corrente deve conter a maioria dos arquivos que é de interesse do processo. Quando é realizada uma referência a um arquivo, o diretório corrente é pesquisado. Caso um arquivo requerido não seja nele encontrado, então o usuário deve especificar um nome de caminho ou passar do diretório corrente para o diretório que mantém esse arquivo. Para mudar de diretório, é fornecida uma chamada de sistema que recebe um nome de diretório como parâmetro e o utiliza para redefinir o diretório corrente. Portanto, o usuário pode mudar seu diretório corrente sempre que quiser. De uma chamada change_directory ( ) para outra, todas as chamadas de sistema open ( ) pesquisam o diretório corrente em busca do arquivo especificado. Observe que o caminho de busca pode, ou não, conter uma entrada especial que represente "o diretório corrente".

O diretório corrente inicial do shell de login de um usuário é designado quando o job do usuário é iniciado ou o usuário faz login. O SO pesquisa o arquivo de contabilidade (ou alguma outra locação predefinida) de modo a encontrar uma entrada para esse usuário (para fins de contabilidade). No arquivo de contabilidade, há um ponteiro para o (ou o nome do) diretório inicial do usuário. Esse ponteiro é copiado para uma variável local do usuário que especifica seu diretório corrente inicial. Outros processos podem ser gerados a partir desse shell. O diretório corrente de qualquer subprocesso é usualmente o diretório corrente do pai quando ele foi gerado.

Os nomes de caminho podem ser de dois tipos: absoluto e relativo. Um nome de caminho absoluto começa na raiz e segue um caminho descendente até o arquivo especificado, fornecendo os nomes de diretório no caminho. Um nome de caminho relativo define um caminho a partir do diretório corrente. Por exemplo, no sistema de arquivos estruturado em árvore da figura anterior, se o diretório corrente for root/spell/mail, então o nome de caminho relativo prt/first referenciará o mesmo arquivo indicado pelo nome de caminho absoluto root/spell/mail/prt/first.

Ao permitir que um usuário defina seus próprios subdiretórios, a organização de diretórios possibilita que o usuário imponha uma estrutura aos seus arquivos. Essa estrutura pode resultar em diretórios separados para arquivos associados a tópicos diferentes (por exemplo, um subdiretório foi criado para conter o texto deste livro-texto) ou tipos de informação diferentes (por exemplo, o diretório programs poderia conter programas-fonte; o diretório bin poderia armazenar todos os binários).

Uma decisão interessante de política, em um diretório estruturado em árvore, diz respeito a como manipular a exclusão de um diretório. Se um diretório estiver vazio, sua entrada no diretório que o contém pode simplesmente ser excluída. No entanto, suponha que o diretório a ser excluído não esteja vazio e contenha vários arquivos ou subdiretórios. Uma entre duas abordagens pode ser adotada. Alguns sistemas não excluirão um diretório, a menos que ele esteja vazio. Portanto, para excluir um diretório, primeiro o usuário precisa excluir todos os arquivos desse diretório. Se existirem subdiretórios, esse procedimento deve ser aplicado a eles recursivamente, para que também possam ser excluídos. Essa abordagem pode resultar em um grande volume de trabalho. Uma abordagem alternativa, como a adotada pelo comando rm do Unix, é fornecer uma opção: quando é feita uma solicitação de exclusão de diretório, todos os arquivos e subdiretórios do diretório também devem ser excluídos. As duas abordagens são bem fáceis de implementar; a escolha é uma questão de política. A última é mais conveniente, mas também mais perigosa, porque uma estrutura de diretório inteira pode ser removida com um comando. Se esse comando for emitido incorretamente, um grande número de arquivos e diretórios precisará ser restaurado (supondo que exista um backup).

Em um sistema de diretórios estruturado em árvore, os usuários podem ser autorizados a acessar, além de seus arquivos, os arquivos de outros usuários. Por exemplo, o usuário B poderia acessar um arquivo do usuário A especificando seus nomes de caminho. O usuário B poderia especificar um nome de caminho absoluto ou relativo. Alternativamente, o usuário B poderia alterar seu diretório corrente para o diretório do usuário A e acessar o arquivo por seus nomes de arquivo.

7.5.4 Diretórios em grafos

Considere dois programadores trabalhando juntos em um projeto. Os arquivos associados a esse projeto podem ser armazenados em um subdiretório, separando-os de outros projetos e arquivos dos dois programadores. Contudo, já que os dois programadores são igualmente responsáveis pelo projeto, ambos querem que o subdiretório esteja em seus próprios diretórios. Nesse caso, o subdiretório comum deve ser compartilhado. Um diretório ou arquivo compartilhado existe no sistema de arquivos em dois (ou mais) locais ao mesmo tempo.

Uma estrutura de árvore proíbe o compartilhamento de arquivos ou diretórios. Um grafo acíclico, isto é, um grafo sem ciclos, permite que os diretórios compartilhem subdiretórios e arquivos e é apresentado na figura a seguir. O mesmo arquivo ou subdiretório pode estar em dois diretórios diferentes. O grafo acíclico é uma generalização natural do esquema de diretório estruturado em árvore.

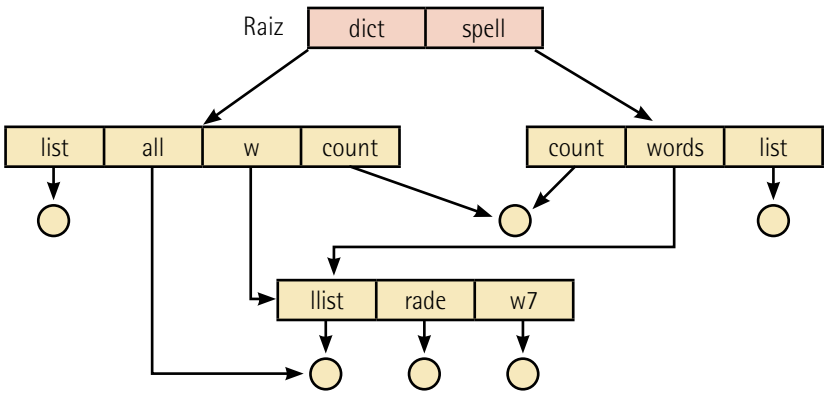

Figura 50 – Estrutura de diretório em grafos

Adaptada de: Silberschatz, Galvin e Gagne (2015, p. 273).

É importante observar que um arquivo (ou diretório) compartilhado não é o mesmo que duas cópias do arquivo. Com duas cópias, cada programador pode visualizar a cópia em vez do original, mas, se um programador alterar o arquivo, as alterações não aparecerão na versão do outro. Com um arquivo compartilhado, existe apenas um arquivo real; portanto, qualquer alteração feita por uma pessoa poderá ser imediatamente vista por outra. O compartilhamento é particularmente importante para os subdiretórios; um novo arquivo criado por uma pessoa aparecerá automaticamente em todos os subdiretórios compartilhados.

7.5.5 Diretório em grafo geral

Um grave problema ao usar uma estrutura de grafo acíclico é assegurar que não existam ciclos. Se começarmos com um diretório de dois níveis e permitirmos que os usuários criem subdiretórios, teremos um diretório estruturado em árvore. É fácil perceber que a simples inclusão de novos arquivos e subdiretórios, em um diretório estruturado em árvore existente, preserva a natureza da estrutura em árvore. No entanto, ao adicionarmos links, a estrutura em árvore é destruída, resultando em uma estrutura de grafo simples.

A principal vantagem de um grafo acíclico é a relativa simplicidade dos algoritmos que percorrem o grafo e determinam quando não há mais referências a um arquivo. Queremos evitar percorrer duas vezes as seções compartilhadas de um grafo acíclico, principalmente por questões de desempenho. Se tivermos acabado de pesquisar um grande subdiretório compartilhado em busca de um arquivo específico, sem encontrá-lo, queremos evitar a pesquisa desse subdiretório novamente; a segunda busca seria um desperdício de tempo.

Se for permitida a existência de ciclos no diretório, também queremos evitar a pesquisa de qualquer componente duas vezes, por motivos de precisão e desempenho. Um algoritmo mal projetado pode resultar em um loop infinito continuamente percorrendo o ciclo e nunca terminando. Uma solução é limitar arbitrariamente o número de diretórios que serão acessados durante uma busca.

Ocorre um problema semelhante ao tentarmos determinar quando um arquivo pode ser excluído. Nas estruturas de diretório em grafo acíclico, um valor igual a 0 na contagem de referências significa que não há mais referências ao arquivo ou diretório, e o arquivo pode ser excluído. No entanto, quando existem ciclos, a contagem de referências pode não ser igual a 0, mesmo se não for mais possível referenciar um diretório ou arquivo. Essa anomalia resulta da possibilidade de autorreferência (ou um ciclo) na estrutura de diretório. Nesse caso, geralmente precisamos usar um esquema de coleta de lixo para determinar quando a última referência foi excluída e o espaço em disco pode ser realocado. A coleta de lixo envolve uma varredura no sistema de arquivos inteiro, marcando tudo que possa ser acessado. Em seguida, uma segunda passagem reúne, em uma lista de espaço livre, tudo que não esteja marcado. Um procedimento de marcação semelhante pode ser usado para assegurar que uma varredura ou pesquisa englobe tudo que existe no sistema de arquivos uma vez e apenas uma vez. A coleta de lixo em um sistema de arquivos baseado em disco é, no entanto, extremamente demorada e, portanto, raramente executada.

A coleta de lixo é necessária apenas em razão da possibilidade de existência de ciclos no grafo. Assim, é muito mais fácil trabalhar com uma estrutura de grafo acíclico. A dificuldade é evitar ciclos conforme novos links são adicionados à estrutura. Como saber quando um novo link completará um ciclo? Há algoritmos que detectam ciclos em grafos; no entanto, eles são computacionalmente dispendiosos, especialmente quando o grafo está armazenado em disco. No caso especial de diretórios e links, um algoritmo mais simples ignoraria os links durante a varredura do diretório. Ciclos são evitados e não há overhead adicional.

7.6 Sistemas de arquivo Windows

Constituem os referidos arquivos:

FAT (FAT32 – Windows 98/95 e Windows 2000)

Armazenado em local fixo para facilitar o acesso aos arquivos, ele é simples e cria uma cópia de segurança deles. Seu tamanho é predefinido em função do volume de informações. A tabela FAT indica qual é o cluster em que o arquivo está armazenado. Não admite definição de permissões em arquivos FAT.

HPFS (Windows NT 3.1, 3.5 e 3.51.)

Organização de arquivos igual à do FAT, porém há a classificação automática no diretório, que é baseada nos nomes dos arquivos. Armazenamento em unidades físicas, e não mais em clusters de partições. O arquivo pode ser composto de dados e atributos de nomeação de arquivos. Esse sistema de arquivos aponta para o que chamamos de FNODE (contém os dados dos arquivos). Ele provê maior eficiência no processamento sequencial.

NTFS

Organiza os arquivos em diretórios como o HPFS. Garante recuperação de dados, identificação e remoção de falhas fatais ao sistema e evita o envio de mensagens de erro ao SO, técnica conhecida como hot fixing. Permite a recuperação de dados em função de se trabalhar com registros que indicam o ponto em que ocorreu a interrupção ou falha, sendo possível recuperar o arquivo.

7.7 Sistemas de arquivos Linux

O sistema de arquivos é responsável pelo gerenciamento das informações que são gravadas em uma determinada partição do disco. O sistema GNU/Linux trabalha com uma grande variedade de sistemas de arquivos. Ele diferencia letras maiúsculas e minúsculas para nomes, arquivos, comandos e diretórios, isto é, ele é case sensitive.

Um arquivo oculto é identificado por um "." no início do nome (por exemplo, .bashrc). Arquivos ocultos não aparecem em listagens normais de diretórios, deve-se usar o comando ls -a para também listar arquivos ocultos.

7.7.1 Permissões de acesso a arquivos e diretórios no Linux

Em um sistema computacional, o compartilhamento de arquivos é bastante desejável para usuários que almejam colaborar e reduzir o esforço requerido para alcançar um objetivo. Assim, SOs devem incorporar o compartilhamento de arquivos, apesar das dificuldades inerentes.

No Linux, existem diferentes tipos de usuários para determinado arquivo. O dono ou owner é o usuário logado no sistema que criou o arquivo ou diretório. Somente o dono pode alterar as permissões de aceso do arquivo. A identificação do dono do arquivo é denominada user id (UID).

Atribuindo diversos usuários a um grupo, possibilita-se que diferentes usuários tenham acesso a um mesmo arquivo. O grupo possui seu próprio identificador de grupo ou group identification (GID). Cada usuário pode participar de um ou mais grupos e conseguirá acessar os arquivos pertencentes aos seus grupos, mesmo que ele não seja o proprietário do arquivo.

Uma terceira categoria de usuários é chamada de outros e contém usuários que não são donos do arquivo e não pertencem ao grupo do arquivo.

Existem três tipos de permissões básicas que podem ser aplicadas ao dono, grupo ou outros usuários: leitura, gravação e execução. A permissão de leitura é identificada com r, do inglês read. No caso de um diretório, o usuário pode listar seu conteúdo.

A permissão de gravação de arquivos é representada pela letra w, da palavra write. Caso seja um diretório, o usuário poderá armazenar arquivos ou criar novos diretórios dentro dele. Assim, para apagar um arquivo ou diretório, é necessário ter a permissão de acesso à gravação.

Se o arquivo for um programa executável, outra permissão é a de execução, que é representada pela letra x. Caso seja um diretório, este possibilitará o acesso ao diretório.

Assim, temos três grupos de permissões aplicáveis a três classes de usuários, o que totaliza nove possibilidades, representadas por bits. O hífen (-) anula uma das três permissões, conforme a posição apresentada.

Existem ainda mais três atributos especiais, associados a um arquivo, o que resulta em 12 bits, conforme (Haeder *et al.*, 2010). Na figura a seguir, temos a estrutura desses bits para o arquivo citado:

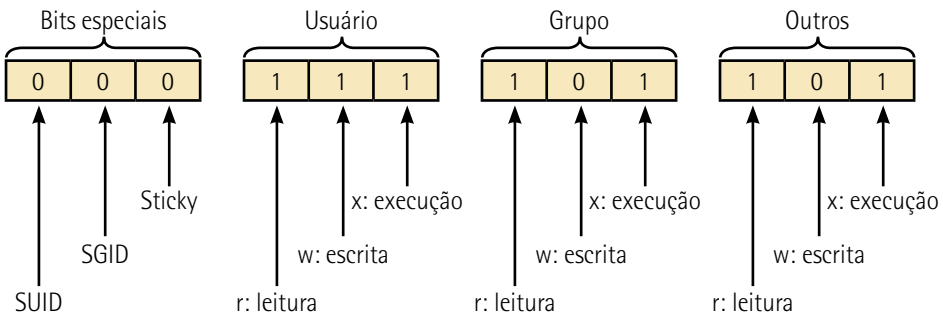

Figura 51 – Estrutura dos bits de permissão para um arquivo

Considere uma situação na qual um usuário navegue até o diretório /usr/bin, digite o comando 'ls -l' e obtenha a seguinte lista de arquivos (mostraremos apenas a parte final da saída, por simplicidade):

-rwxr-xr-x 1 root root 186664 ago 16 2019 zipinfo

-rwxr-xr-x 1 root root 89488 abr 21 2017 zipnote

-rwxr-xr-x 1 root root 93584 abr 21 2017 zipsplit

-rwxr-xr-x 1 root root 2206 dez 13 2019 zless

-rwxr-xr-x 1 root root 1842 dez 13 2019 zmore

-rwxr-xr-x 1 root root 4553 dez 13 2019 znew

Observe que, na primeira coluna da saída, temos um conjunto de letras e de símbolos, como -rwxr-xr-x. Essas são as informações de permissão do arquivo.

7.7.2 Journaling

O journaling é uma área do SO responsável pelos registros de informações de arquivos e das respectivas atividades básicas que foram realizadas em um determinado arquivo. O sistema de journaling armazena qualquer operação que será feita no disco em uma área especial chamada journal ou diário,

em português, de forma que se acontecer algum problema durante alterações no disco, ele poderá voltar ao estado anterior do arquivo ou finalizar a operação.

O journaling permite o rastreamento das alterações que não foram confirmadas pelo sistema de arquivos, de modo a registrá-las em uma estrutura denominada log de dados. Com isso, oferece alta tolerância a falhas. Há o journal físico, que faz uma cópia dos arquivos depois de gravados no sistema de arquivos principal e o journal lógico, responsável por gravar os metadados que podem sofrer as alterações, como leitura/gravação e atualização de arquivos.

Um conjunto de operações que executam uma tarefa específica é denominado transação. Quando uma transação é armazenada no diário, considera-se que ela está confirmada. Enquanto isso, as entradas do diário relacionadas com a transação são reexecutadas nas estruturas reais do sistema de arquivos. Conforme as alterações são realizadas, um ponteiro é atualizado para indicar quais ações foram concluídas e quais ainda estão incompletas. Quando uma transação confirmada em sua totalidade é concluída, ela é removida do diário. O diário pode ser entendido como um buffer circular e pode estar em uma seção separada do sistema de arquivos ou, até mesmo, em um eixo separado do disco. É mais eficiente, porém mais complexo, tê-lo sob cabeçotes de leitura-gravação separados, diminuindo assim os tempos de disputa e busca do cabeçote.

Com o journal, o sistema de arquivos tem melhoria na disponibilidade e maior tolerância a falhas. De acordo com Silva (2020b), um exemplo dessas melhorias pode ser observado após uma falha de energia, quando o journal será analisado durante a montagem do sistema de arquivos e todas as operações que estavam sendo feitas no disco serão verificadas e podem ser finalizadas ou desfeitas em função do estado da operação. Considerando que o retorno do servidor é praticamente imediato, é possível realizar um rápido reestabelecimento dos serviços da máquina, melhorando a disponibilidade.

Outro caso que pode ser evitado através de journaling é o de inconsistências no sistema de arquivos do servidor após a situação citada, fazendo o servidor ficar em estado single user e esperando pela intervenção do administrador.

Sistemas de arquivos e diretórios no Linux

As primeiras versões do Linux foram programadas com um sistema de arquivos compatível com o SO Minix, para facilitar a troca de dados com o sistema de desenvolvimento Minix. Entretanto, esse sistema possuía restrições severas de limites de nome de arquivo de 14 caracteres e tamanho máximo do sistema de arquivos de 64 MB. O Minix foi superado por um novo sistema batizado como sistema de arquivos estendido ou extended file system (EXT).

Um reprojeto posterior, para melhorar o desempenho e a escalabilidade e adicionar alguns recursos ausentes, levou ao segundo sistema de arquivos estendido (ext2). Um novo desenvolvimento adicionou recursos de diário e o sistema foi renomeado como terceiro sistema de arquivos estendido (ext3). Os desenvolvedores do kernel do Linux estão trabalhando na ampliação do ext3 com recursos de sistemas de arquivos modernos, tais como as extensões. Ele é chamado quarto sistema de arquivos estendido (ext4).

Lembrete

Inicialmente, as versões do Linux eram programadas com um sistema de arquivos compatível com o SO Minix, para facilitar a troca de dados com o sistema de desenvolvimento Minix.

A seguir estão detalhadas algumas características de sistemas de arquivos.

EXT2

- Conhecido como second extended file system.
- Utiliza blocos do mesmo tamanho para armazenar os arquivos.
- Trabalha com partições de até 4 terabytes.
- O tamanho máximo do nome do arquivo é de 255 caracteres.
- Reserva normalmente 5% dos blocos para o superusuário (root).
- Não possui journaling e foi substituído pelo ext3.

EXT3

- Permite utilizar o sistema de cotas.
- Pode trabalhar com blocos de 1, 2 e 4 kilobytes.
- O tamanho máximo da partição é de 16 terabytes.
- O tamanho máximo dos arquivos é de 2 terabytes.

EXT4

- Pode facilmente ser convertido para o formato ext4.
- Possui melhorias no desempenho e na capacidade de armazenamento.
- Disponibiliza o recurso de journaling, podendo este ser habilitado ou não.
- O tamanho máximo de cada arquivo é de 16 terabytes.

ReiserFS

- Possui suporte ao journaling.
- Trabalha com blocos de 4 kilobytes.
- O tamanho máximo da partição é de 1 ectabyte.
- O tamanho máximo do arquivo é de 1 ectabyte.
- Utiliza a estrutura de dados para armazenamento conhecida como árvore B+, e os dados são armazenados em unidades chamadas de folhas.

XFS

- Permite utilizar o sistema de cotas de usuários e de grupos.
- Trabalha com blocos de até 64 kilobytes.
- Possui suporte ao journaling.
- O tamanho máximo da partição é de 16 terabytes (32 bits) e 8,58 ectabytes (64 bits), e o tamanho máximo dos arquivos é de 16 terabytes no Linux 2.4 e de 8,58 ectabytes no Linux 2.6, estando com o endereçamento de 64 bits acionado.

O sistema ext3 utiliza um mecanismo equivalente à localização dos blocos de dados pertencentes a um arquivo específico, armazenando ponteiros de blocos de dados em blocos indiretos em todo o sistema de arquivos com até três níveis de endereçamento indireto. Os arquivos de diretório são armazenados em disco como arquivos normais. Entretanto, o conteúdo é interpretado diferentemente. Cada bloco em um arquivo de diretório consiste em uma lista encadeada de entradas, e cada entrada contém o tamanho da entrada, o nome de um arquivo e o número de inode do inode ao qual essa entrada se refere.

Para maximizar o desempenho, sempre que possível o SO deve tentar executar operações de I/O em grandes blocos agrupando solicitações de I/O fisicamente adjacentes. O agrupamento reduz o overhead por solicitação em que incorrem os drivers de dispositivos, discos e hardware do controlador de discos. Um tamanho de solicitação de I/O correspondente a um bloco é pequeno demais para manter um bom desempenho e, assim, o ext3 usa políticas de alocação projetadas para inserir blocos logicamente adjacentes de um arquivo em blocos fisicamente adjacentes em disco, para que ele possa submeter uma solicitação de I/O para vários blocos de disco como uma única operação.

A política de alocação do ext3 funciona da seguinte forma: como no FFS, um sistema de arquivos ext3 é particionado em múltiplos segmentos. No ext3, eles são chamados grupos de blocos. O FFS usa o conceito semelhante de grupos de cilindros, em que cada grupo corresponde a um único cilindro de um disco físico.

Ao alocar um arquivo, o ext3 deve primeiro selecionar o grupo de blocos desse arquivo. Para blocos de dados, ele tenta alocar o arquivo ao grupo de blocos em que o inode do arquivo foi alocado. Para alocações de inodes, é selecionado o grupo de blocos em que o diretório-pai do arquivo reside, no caso de arquivos que não sejam de diretório. Arquivos de diretório não são armazenados juntos, permanecendo dispersos em todos os grupos de blocos disponíveis. Essas políticas são projetadas não apenas para manter informações relacionadas no mesmo grupo de blocos, mas para distribuir a carga do disco entre os grupos de blocos, reduzindo a fragmentação de quaisquer áreas do disco. A figura a seguir ilustra a política de alocação do ext3 e do ext4.

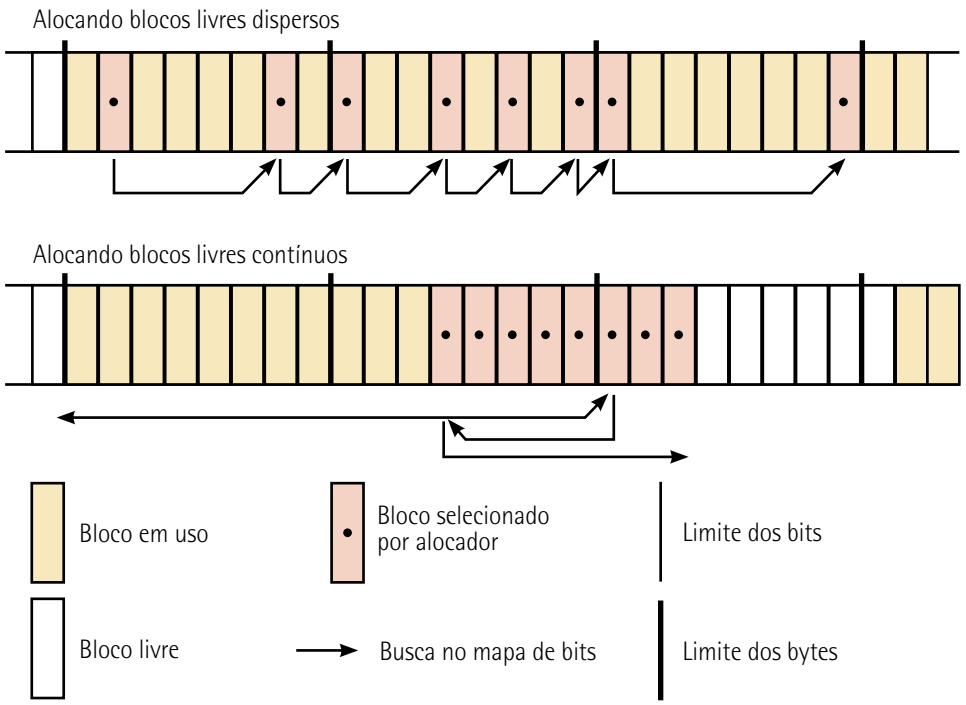

Figura 52 – Política de alocação de blocos no ext3 e ext4

Adaptada de: Silberschatz, Galvin e Gagne (2015, p. 432).

Em um grupo de blocos, o ext3 tenta manter, se possível, as alocações fisicamente contíguas reduzindo a fragmentação. Ele mantém um mapa de bits de todos os blocos livres em um grupo de blocos. Ao alocar os primeiros blocos a um novo arquivo, ele começa procurando por um bloco livre a partir do início do grupo de blocos. Ao estender um arquivo, ele continua a busca a partir do bloco alocado mais recentemente ao arquivo. A busca é executada em dois estágios. Inicialmente, o ext3 procura por um byte totalmente livre no mapa de bits; caso não consiga encontrar um, ele buscará qualquer bit livre.

Quando é identificado um bloco livre, a busca retrocede até que um bloco alocado seja encontrado. Quando um byte livre é encontrado no mapa de bits, essa busca na direção inversa impede que o ext3 deixe uma lacuna entre o bloco alocado mais recentemente no byte anterior diferente de zero e o byte zero encontrado. Uma vez que o próximo bloco a ser alocado tenha sido encontrado pela busca do bit ou do byte, o ext3 estende a alocação para até oito blocos à frente e pré-aloca esses blocos adicionais ao arquivo. Essa pré-alocação ajuda a reduzir a fragmentação durante gravações intercaladas em arquivos

separados e também reduz o custo da CPU referente à alocação de disco por meio da alocação de múltiplos blocos simultaneamente. Os blocos pré-alocados são devolvidos ao mapa de bits de espaço livre quando o arquivo é fechado.

Observação

A moderna tecnologia de drives de disco compacta setores no disco em diferentes densidades e, portanto, com diversos tamanhos de cilindro, dependendo da distância do cabeçote ao centro do disco. Portanto, grupos de cilindros de tamanho fixo não correspondem necessariamente à geometria do disco.

7.7.3 Diretórios do sistema (FHS)

Como existem diversas distribuições disponíveis para o Linux, foi necessário realizar uma padronização nos diretórios encontrados. Essa padronização é conhecida como filesystem hierarchy standard (FHS).

Segundo Negus (2014), o FHS é utilizado por aproximadamente 99% dos Linux, sendo mantido pela Free Standard Groups, que possui engenheiros de organizações como IBM, Red Hat, Dell e HP.

Essa estrutura está dividida de forma hierárquica, em que cada diretório possui uma finalidade diferente. O diretório raiz do sistema é representado por uma barra normal (/). Abaixo do diretório raiz, são indicados os subdiretórios que organizam os arquivos no sistema, conforme exibido na figura a seguir.

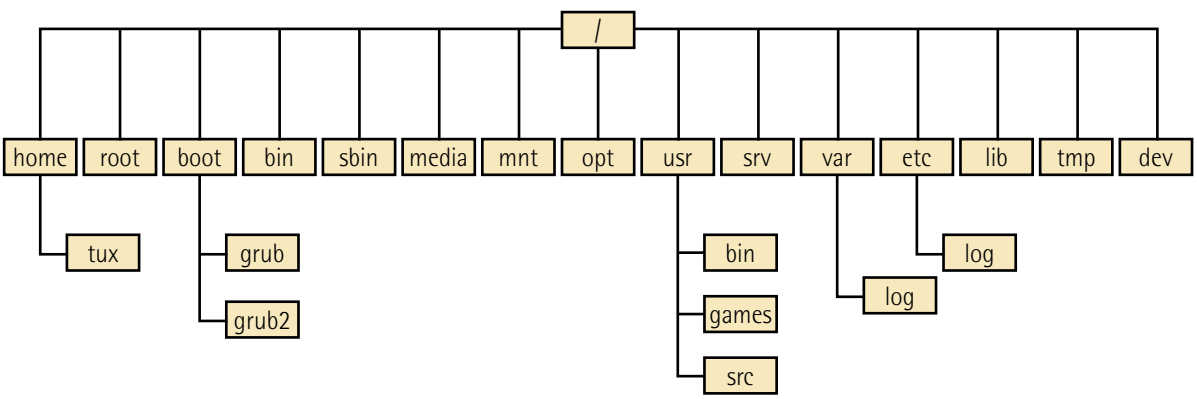

Figura 53 – Estrutura do FHS

Fonte: Alves (2018, p. 185).

O diretório raiz é o diretório principal do Linux, onde estão todos os seus diretórios. Ele é representado por uma "/". Nele estão localizados outros diretórios como o /bin, /sbin, /usr, /usr/local, /mnt, /tmp, /var, /home, entre outros. Estes são denominados subdiretórios, pois estão dentro do diretório "/".

De acordo com Silva (2020b), o GNU/Linux possui a seguinte estrutura básica de diretórios organizados segundo o FHS.

O diretório /bin possui os arquivos de programas do sistema que são usados com frequência pelos usuários. Já o diretório /boot tem arquivos necessários para a inicialização do sistema. No diretório /media, estão os arquivos montagem de dispositivos diversos do sistema, tais como rede, pendrives.

No diretório /dev são encontrados os arquivos usados para acessar dispositivos de E/S ou periféricos existentes no computador. O diretório /etc tem os arquivos de configuração para o desktop. O diretório /root possui os arquivos do usuário root ou superusuário, o login que não possui restrições de segurança. Já no diretório /home, estão os diretórios contendo os arquivos dos usuários, e cada usuário terá um subdiretório. As bibliotecas compartilhadas pelos programas do sistema e módulos do kernel do SO estão no diretório /lib. /

O diretório /sys contém o sistema de arquivos do kernel e é utilizado por diversos programas para realizar a leitura, verificar configurações do sistema ou modificar o funcionamento de dispositivos do sistema através da alteração em seus arquivos.

O diretório /usr contém os arquivos da maior parte de seus programas e normalmente apenas é acessado para leitura. Os arquivos de armazenamento temporários gerados pelos programas são salvos no diretório /tmp.

Os arquivos mais utilizados pelos programas do sistema, cache e spool de impressora são armazenados no diretório /var. Existem programas usados pelo superusuário que funcionam para administrar e controlar o sistema e estão disponíveis no diretório /sbin.

Diretórios "virtuais" /proc e /sys e seus arquivos

Os dispositivos de hardware, com exceção da interface de rede, são representados por um arquivo como: HD, terminais de acesso, portas de comunicação, placa de vídeo, placa de som etc.

Esses arquivos ficam armazenados no diretório /dev/, onde são criados durante o momento do boot do sistema. Existem dois diretórios "virtuais" que também são criados no momento da inicialização do sistema. São eles:

- /proc e o /sys/. Esses dois diretórios não são criados na instalação do sistema. Para controlar os dispositivos de hardware em funcionamento e todos os processos ativos do sistema, o kernel utiliza ambos os diretórios.

Esses arquivos são solicitados por alguns comandos do sistema, por exemplo, o comando lsmod, que informa os módulos carregados. São exemplos:

- **/proc/interrupts**: informa as IRQs em uso por cada periférico.

- **/proc/bus**: contém informações específicas do barramento.
- **/proc/cpuinfo**: possui informações do processador utilizado (modelo, fabricante etc.).
- **/proc/devices**: informa os drivers de dispositivos configurados no kernel que estão em execução.
- **/proc/dma**: destaca os canais de acesso à memória que estão em uso.
- **/proc/diskstats**: relata as estatísticas de E/S dos dispositivos de bloco.
- **/proc/filesystems**: acentua os sistemas de arquivos que possuem suporte do kernel.
- **/proc/ioports**: informa as portas de E/S que estão sendo utilizadas.
- **/proc/meminfo**: destaca o uso, pelo sistema, de memória RAM, buffer, cache, entre outros.
- **/proc/modules**: relata os módulos que foram carregados pelo kernel.
- **/proc/partitions**: acentua os HDs e suas devidas partições reconhecidas pelo sistema.
- **/proc/swaps**: informa o tamanho da memória swap utilizada.

No diretório /sys também existem outros arquivos referentes ao hardware, em que os valores podem ser alterados. São exemplos:

- **/sys/block**: informações dos dispositivos de blocos do HD, CD, DVD, entre outros.
- **/sys/bus**: informações dos barramentos que possuem suporte do kernel.
- **/sys/class**: organização dos tipos de dispositivos que possuem suporte do sistema em subdiretórios.
- **/sys/module**: módulos carregados na memória pelo kernel. Podemos habilitá-los/desabilitá-los usando o comando modprob. Sempre que um dispositivo é adicionado, um arquivo é criado nesse diretório e gerenciado pelo programa udev.

7.8 Sistemas de arquivos MacOS

O sistema de arquivos do Mac OS a partir da versão iOS 10.3 é o Apple File System, que substituiu o HFS Plus. O Apple File System oferece um sistema de arquivos melhorado com diversas características, como compartilhamento de espaço, snapshots, que é o registro do estado do sistema, da aplicação ou de arquivos em determinado instante de tempo.

Ele é dividido em duas camadas: a de contêiner e a do sistema de arquivos (Apple, 2020). A camada de contêiner organiza a informação da camada de sistema de arquivo e armazena as informações de forma consolidada, como snapshots do volume.

A camada de sistema de arquivos é composta de estrutura de dados que armazenam informações, como a estrutura de diretórios.

Exemplo de aplicação

Considere que um arquivo produzido no sistema MacOS X por um processador de textos tem o nome do processador de texto como seu criador. Quando o usuário clicar duas vezes no ícone desse arquivo no explorador de arquivos, o processador de texto é invocado de forma automática e o arquivo é carregado, pronto para ser editado.

Outra característica do sistema de arquivos Mac OS X é que, na primeira utilização de um disco no sistema, tanto em tempo de inicialização quanto na execução do sistema, o Mac OS X procura por um sistema de arquivos no dispositivo. Então, quando o encontra, ele monta automaticamente o sistema de arquivos sob o diretório /Volumes, adicionando um ícone de pasta rotulado com o nome do sistema de arquivos (como armazenado no diretório do dispositivo). O usuário pode clicar no ícone e exibir o sistema de arquivos recém-montado.

Saiba mais

Para os desenvolvedores de aplicativos, a Apple disponibiliza uma documentação completa sobre o sistema de arquivos e utilização de APIs de alto nível para usar as funções do sistema de arquivos da Apple, sem precisar de alterações no código do aplicativo.

APPLE. *About Apple file system*. 2020. Disponível em: https://shre.ink/nOQm. Acesso em: 25 set. 2023.

8 SEGURANÇA E PROTEÇÃO NOS SISTEMAS OPERACIONAIS

Neste tópico, abordaremos o uso e a administração de SOs de rede do ponto de vista da segurança e da proteção.

De acordo com Silberschatz, Galvin e Gagne (2015), a proteção está ligada a um mecanismo destinado ao controle do acesso de programas e ao controle de processos ou de usuários aos recursos definidos por um sistema de computação. Esse mecanismo, naturalmente, deve ser implementado de forma sólida e incontornável. Adicionalmente, tal implementação precisa ser feita de modo que haja uma forma padronizada de solicitar e conceder (ou não) o acesso apenas quando necessário, ou de suspender (ou abortar) o processo solicitante, quando não for possível conceder tal acesso.

Um sistema de proteção robusto será ineficaz se a autenticação de usuários for comprometida ou se usuários não autorizados conseguirem, de alguma forma, invadi-lo. Os recursos computacionais devem ser protegidos contra acesso não autorizado, destruição ou alteração maliciosa e introdução acidental

de inconsistências. Tais recursos incluem as informações armazenadas no sistema (tanto dados quanto código) e os componentes de hardware das várias máquinas e da rede que as interliga internamente e com o exterior da empresa.

A proteção está relacionada a um problema estritamente interno, que pode ser sintetizado como conceder acesso controlado a dados e software armazenados em um sistema de computação. Por sua vez, a segurança requer não apenas um sistema de proteção adequado, mas necessita que se considere o ambiente externo ao operado pelo sistema computacional.

8.1 Finalidades da proteção em um sistema operacional

Existem diversas razões para investimentos na segurança de um SO. Uma delas é a proteção do sistema e a garantia de que ele continuará operando de forma adequada. Outro aspecto importante é assegurar que o acesso e o uso da informação sejam realizados de acordo com políticas previamente estabelecidas pelas empresas (Silberschatz; Galvin; Gagne, 2015).

Em alguns casos, a violação de segurança acontece de modo não intencional: um usuário com pouco conhecimento a respeito de um sistema computacional pode causar instabilidades e erros se tiver permissões de manipular elevada quantidade de dados e realizá-la de forma equivocada. É importante ter mecanismos para garantirmos que esse tipo de situação não ocorrerá.

Um sistema orientado à proteção fornece meios para diferenciar o uso autorizado do uso não autorizado. O papel da proteção em um sistema de computação é o fornecimento de um mecanismo para a imposição das políticas e das regras que governam o uso de recursos. De acordo com Silva (2020b), essas políticas e regras podem ser estabelecidas de várias maneiras:

- pela introdução de itens no projeto do sistema;
- pelo gerenciamento do sistema;
- pelas atitudes de proteção dos usuários individuais em relação aos seus próprios arquivos e programas.

Um sistema de proteção deve possuir a flexibilidade necessária para permitir a imposição de uma variedade de políticas, posto que as políticas de uso de recursos podem ser alteradas com o tempo.

Outro aspecto importante que devemos ter em mente, no âmbito da análise deste tópico, é a diferença conceitual existente entre mecanismo e política. Um mecanismo descreve a maneira como algo é feito, enquanto uma política define o que deve ser feito ou não, sem se preocupar com o modo como algo é feito.

Essas duas ideias, política e mecanismo, são complementares, sendo que o objetivo geral de um administrador de sistemas é a definição de políticas com foco na descrição dos mecanismos, que devem apresentar flexibilidade.

8.2 Segurança

De acordo com Silva (2023), o SO desempenha um papel crítico na operação de um computador e, como existem vulnerabilidades nele, este também é alvo de diversos ataques cibernéticos. Dado que o SO controla a utilização do hardware, a segurança dele gera um efeito cascata sobre a segurança geral de um computador.

Um administrador codifica um SO ao modificar a configuração padrão para torná-lo mais seguro em relação a ameaças externas. Esse processo inclui a remoção de programas e serviços desnecessários. Outro requisito crítico de codificação de SOs é a aplicação de patches e atualizações de segurança. Patches e atualizações de segurança são correções que as empresas liberam para tentar reduzir a vulnerabilidade e corrigir falhas em seus produtos.

Deve-se ressaltar que segurança total é um ideal inatingível, pois sempre pode ocorrer alguma situação imprevisível capaz de comprometer a segurança. No entanto, um parque computacional será seguro desde que ele que seja acessado e utilizado sempre de acordo com o planejado.

Observação

Os serviços de segurança são fornecidos por uma camada de protocolo de comunicação de sistemas abertos e visam garantir a segurança adequada dos sistemas e das operações de transferências de dados (Stallings, 2015). Eles objetivam a implementação de políticas de segurança por meio da instalação de mecanismos de segurança e podem ser divididos nas categorias a seguir:

- autenticação;
- controle de acesso;
- confidencialidade dos dados;
- integridade dos dados;
- irretratabilidade.

Podemos imaginar que a segurança dos SOs está muito maior nos dias de hoje do que foi no passado. Ainda que isso seja parcialmente verdade, dado que houve melhora substancial na complexidade da estrutura de segurança dos SOs atuais, não há como dizer que estamos totalmente seguros.

Na tabela a seguir, é apresentada a quantidade histórica de vulnerabilidades por SO no período de 1999 a 2022, segundo apuração do site CVEDetails.com.

Tabela 5 – Quantidade de vulnerabilidades identificadas por sistema operacional no período de 1999 a 2022

Nome do SO	Nome do fabricante	Tipo de produto	Número de vulnerabilidades históricas
1 Debian Linux	Debian	SO	7489
2 Android	Google	SO	4902
3 Fedora	Fedora project	SO	4108
4 Ubuntu Linux	Canonical	SO	3709
5 Mac OS X	Apple	SO	3101
6 Linux Kernel	Linux	SO	3034
7 Windows 10	Microsoft	SO	3016
8 IPhone OS	Apple	SO	2863
9 Windows Server 2016	Microsoft	SO	2786
10 Windows Server 2008	Microsoft	SO	2445

Adaptada de: https://tinyurl.com/ydxzxxkb. Acesso em: 25 set. 2023.

Podemos observar que a distribuição Debian GNU/Linux aparece em primeiro lugar com relação ao número de vulnerabilidades identificadas, ainda que boa parte delas tenha sido corrigida. A plataforma Android, um dos SOs mais populares para aparelhos de telefonia celular, aparece em segundo lugar. Evidentemente, apenas o número de vulnerabilidades não é suficiente para dizermos o quão seguro ou inseguro um SO é, pois também precisamos analisar a gravidade da vulnerabilidade e o tempo que ela levou para ser corrigida. Contudo, o extenso número mostra como a questão da segurança ainda é complexa.

Lembrete

A sigla CVE representa as vulnerabilidades e exposições comuns ou common vulnerabilities and exposures, em inglês. Trata-se da realização da catalogação e disponibilização de todas as vulnerabilidades de cibersegurança reportadas.

8.2.1 Gerenciador de patches

Os patches representam atualizações de código que os fabricantes disponibilizam para evitar que um vírus ou um worm recém-descoberto realize um ataque de exploração bem-sucedido. Periodicamente, os fabricantes combinam patches e atualizações em uma aplicação completa de atualização chamada de service pack. Muitos ataques devastadores de vírus poderiam ter sido muito menos graves se mais usuários tivessem baixado e instalado o service pack mais recente.

O Windows verifica regularmente o site *Windows Update* para obter atualizações de alta prioridade e que podem ajudar a proteger o computador contra as mais recentes ameaças. Essas atualizações

incluem atualizações de segurança, atualizações críticas e service packs. Dependendo da configuração escolhida, o Windows baixa e instala, automaticamente, todas as atualizações de alta prioridade que o computador precisa ou notifica o usuário conforme elas estejam disponíveis.

Algumas organizações testam um patch antes de implantá-lo em toda a organização. Elas usariam um serviço para gerenciar patches localmente, em vez de usar o serviço de atualização on-line do fornecedor. Dessa forma, um serviço de patches automáticos provê aos administradores um ambiente mais controlado.

8.3 Modelo de segurança no Linux

De acordo com Silberschatz, Galvin e Gagne (2015), o modelo de segurança do Linux está intimamente relacionado com os mecanismos de segurança típicos do Unix. As preocupações de segurança podem ser classificadas em dois grupos: autenticação e controle de acesso.

A autenticação representa a garantia de que nenhum usuário pode acessar o sistema sem primeiro provar que tem direitos de entrada. O controle de acesso se refere ao fornecimento de um mecanismo que verifique se um usuário tem o direito de acessar determinado objeto e impedir o acesso a objetos quando necessário.

8.3.1 Autenticação

A autenticação no Unix tem sido tipicamente executada por meio do uso de um arquivo de senhas publicamente legível. Uma senha de usuário é combinada com um valor "salt" aleatório, e o resultado é codificado com uma função de transformação unidirecional e armazenado no arquivo de senhas. O uso da função unidirecional significa que a senha original não pode ser deduzida do arquivo de senhas, exceto por tentativa e erro. Quando um usuário apresenta uma senha ao sistema, ela é recombinada com o valor salt armazenado no arquivo de senhas e passa pela mesma transformação unidirecional. Se o resultado coincidir com o conteúdo do arquivo de senhas, a senha será aceita.

Historicamente, as implementações desse mecanismo no Unix têm apresentado vários problemas. As senhas eram, com frequência, limitadas a oito caracteres, e o número de valores salt possíveis era tão baixo que um invasor poderia facilmente combinar um dicionário de senhas comumente utilizadas com cada valor salt possível e ter boa chance de obter uma ou mais senhas no arquivo de senhas, ganhando acesso não autorizado a quaisquer contas, que ficariam então comprometidas. Extensões ao mecanismo de senhas têm sido introduzidas de modo a manterem a senha criptografada secreta em um arquivo que não seja legível publicamente, que permita senhas mais longas ou que use métodos mais seguros em sua codificação.

Outros mecanismos de autenticação têm sido introduzidos para limitar os períodos durante os quais um usuário é autorizado a se conectar ao sistema. Além disso, existem mecanismos para distribuir informações de autenticação a todos os sistemas relacionados em uma rede.

Um novo mecanismo de segurança foi desenvolvido por fornecedores do Unix para resolver problemas de autenticação. O sistema de módulos de autenticação conectáveis (PAM – pluggable authentication modules) é baseado em uma biblioteca compartilhada que pode ser usada por qualquer componente do sistema que precise autenticar usuários. Uma implementação desse sistema está disponível no Linux. O PAM permite que módulos de autenticação sejam carregados sob demanda como especificado em um arquivo de configurações com abrangência em todo o sistema. Se um novo mecanismo de autenticação for adicionado em uma data posterior, ele poderá ser adicionado ao arquivo de configurações, e todos os seus componentes poderão usá-lo imediatamente. Os módulos do PAM podem especificar métodos de autenticação, restrições de contas, funções de configuração de sessões e funções de mudança de senhas (de modo que, quando os usuários mudarem suas senhas, todos os mecanismos de autenticação necessários possam ser atualizados imediatamente).

8.3.2 Controle de acesso

O controle de acesso em Unix, inclusive no Linux, é executado por meio do uso de identificadores numéricos exclusivos. Um identificador de usuário (UID) reconhece um único usuário ou um único conjunto de direitos de acesso. Um identificador de grupo ou group identification (GID) é um identificador adicional que pode ser usado para reconhecer direitos pertencentes a mais de um usuário.

O controle de acesso é aplicado a vários objetos no sistema. Cada arquivo nele disponível é protegido pelo mecanismo de controle de acesso padrão. Além disso, outros objetos compartilhados, tais como seções de memória compartilhada e semáforos, empregam o mesmo sistema de acesso.

Cada objeto no Unix, sob controle de acesso de usuários e grupos, tem um UID e um GID exclusivos associados a ele. Processos de usuário também têm um UID exclusivo, mas podem ter mais de um GID. Se o UID de um processo coincidir com o UID de um objeto, então o processo tem direitos de usuário ou direitos de proprietário sobre esse objeto. Se os UIDs não coincidirem, mas algum GID do processo coincidir com o GID do objeto, então direitos de grupo serão conferidos; caso contrário, o processo terá direitos universais sobre o objeto.

O Linux executa o controle de acesso atribuindo aos objetos uma máscara de proteção que especifica as modalidades de acesso – leitura, gravação ou execução – que devem ser concedidas a processos com acesso de proprietário, de grupo ou universal. Assim, o proprietário de um objeto pode ter acesso total de leitura, gravação e execução para um arquivo; outros usuários de determinado grupo podem receber acesso de leitura, mas ter negado o acesso de gravação; e o restante das pessoas pode não receber nenhum tipo de acesso.

A única exceção é o UID raiz privilegiado. Um processo com esse UID especial recebe acesso automático a qualquer objeto no sistema, ignorando verificações de acesso normais. Tais processos também recebem permissão para executar operações privilegiadas, tais como a leitura de qualquer memória física ou a abertura de sockets de rede reservados. Esse mecanismo permite que o kernel impeça usuários comuns de acessarem esses recursos: a maioria dos recursos internos importantes do kernel é implicitamente de propriedade do UID raiz. O Linux implementa o mecanismo setuid padrão do Unix.

Esse mecanismo permite que um programa seja executado com privilégios diferentes daqueles referentes ao usuário que o está executando. Por exemplo, o programa lpr (que submete um job a uma fila de impressão) tem acesso às filas de impressão do sistema, mesmo se o usuário que o estiver executando não tenha. A implementação do setuid no Unix diferencia os UIDs real e efetivo de um processo. O UID real é o do usuário que está executando o programa; o UID efetivo é o do proprietário do arquivo. No Linux, esse mecanismo é ampliado de duas maneiras.

Em primeiro lugar, o Linux implementa o mecanismo saveduser-id da especificação POSIX, que permite que um processo abandone e readquira seu UID efetivo repetidamente. Por motivos de segurança, um programa talvez queira executar a maioria de suas operações de modo seguro, abandonando os privilégios concedidos por seu status setuid; mas ele pode pretender executar operações selecionadas com todos os seus privilégios. As implementações padrão do Unix disponibilizam esse recurso somente por permuta entre os UIDs real e efetivo. Quando isso é feito, o UID efetivo anterior é lembrado, mas o UID real do programa nem sempre corresponde àquele do usuário que está executando o programa. UIDs salvos permitem que um processo defina seu UID efetivo como seu UID real e, então, retorne ao valor anterior efetivo sem ter que modificar o UID real em momento algum.

A segunda melhoria fornecida pelo Linux é a incorporação de uma característica dos processos que concede apenas um subconjunto dos direitos do UID efetivo. As propriedades de processos fsuid e fsgid são usadas quando são concedidos direitos de acesso a arquivos. A propriedade adequada é estabelecida sempre que o UID ou GID efetivo é definido.

No entanto, o fsuid e o fsgid podem ser definidos independentemente dos ids efetivos, permitindo que um processo acesse arquivos em nome de outro usuário sem, de forma alguma, assumir a identidade desse outro. Especificamente, processos de servidor podem usar esse mecanismo a fim de servir arquivos para determinado usuário sem ficarem vulneráveis ao encerramento ou suspensão por esse usuário.

Para concluir, o Linux fornece um mecanismo para a transmissão flexível de direitos de um programa para outro – um mecanismo que se tornou comum em versões modernas do Unix. Quando um socket de rede local é estabelecido entre dois processos no sistema, um desses processos pode enviar ao outro o descritor de arquivo de um de seus arquivos abertos; o outro processo recebe um descritor de arquivo duplicata do mesmo arquivo.

Esse mecanismo permite que um cliente passe, seletivamente, o acesso a um único arquivo para algum processo de servidor sem conceder a ele nenhum outro privilégio. Por exemplo, não é mais necessário que um servidor de impressão tenha que ler todos os arquivos de um usuário que submeta um novo job de impressão. O cliente da impressão pode simplesmente passar ao servidor os descritores de arquivo de quaisquer arquivos a serem impressos, negando ao servidor acesso a qualquer dos outros arquivos do usuário.

8.4 Introdução à segurança da Apple

Segundo Apple (2020), a empresa atribui à segurança o centro de suas plataformas e trabalha a serviço do objetivo final de manter informações pessoais seguras. Ela criou arquiteturas de segurança que atendem aos requisitos especiais de dispositivos móveis e computadores.

Existem recursos e serviços de segurança, como a criptografia do dispositivo com base no hardware, o Face ID e o Touch ID, que utilizam a biometria facial e impressão digital como formas de autenticação, respectivamente. Também é disponibilizada a segurança de serviços utilizados para identificação, gerenciamento de senhas, pagamentos via carteiras digitais e inclusive a procura de equipamentos perdidos.

Outro ponto é a preocupação da Apple com a privacidade, a qual é considerada como um direito humano fundamental, oferecendo diversos controles e opções para que os próprios usuários sejam responsáveis pela decisão de como, quais e quando os aplicativos podem utilizar as informações pessoais do usuário.

Saiba mais

Para os desenvolvedores de aplicativos, a Apple disponibiliza a documentação completa acerca da segurança da plataforma Apple, que abrange esta documentação e versões dos SOs: iOS, iPadOS, macOS, tvOS e watchOS.

APPLE. *Introdução à segurança da plataforma Apple*. 2022. Disponível em: https://shre.ink/nOQq. Acesso em: 25 set. 2023.

Resumo

Nesta unidade, observamos que o sistema de arquivos é a parte mais visível do SO do ponto de vista dos usuários e que o compartilhamento de arquivos é fundamental para que os usuários trabalhem de forma colaborativa.

Tomamos ciência de que o problema da segurança de SOs é que, em um mundo no qual a tecnologia é cada vez mais presente e essencial, existem incentivos e até facilidades para que as vulnerabilidades sejam exploradas. Quanto mais popular um SO se torna, mais pessoas buscam identificar potenciais vulnerabilidades e furos de segurança.

Por fim, entendemos que outros problemas se referem à própria indústria de software e à natureza do modo como esses projetos são gerenciados. Um SO não é apenas um único software, mas uma coleção enorme de programas com finalidades específicas. A pressão para a rápida produção de um código e a introdução de novas funcionalidades pode levar à escrita de um código-fonte que possibilite a existência de vulnerabilidades em um sistema. É improvável que esse cenário venha a mudar em pouco tempo, e, por mais que medidas de segurança sejam tomadas, atacantes sempre buscarão os pontos fracos de um sistema, isto é, suas vulnerabilidades.

Exercícios

Questão 1. (Inaz do Pará 2018, adaptada) No Linux, os discos e suas respectivas partições são montados na estrutura de diretórios, e o diretório que representa a partição do disco é chamado de ponto de montagem. Qual é a alternativa que indica o diretório onde estão localizados os arquivos executáveis de comandos básicos do SO utilizados com frequência pelo usuário?

A) /media

B) /dev

C) /boot

D) /bin

E) /etc

Resposta correta: alternativa D.

Análise das alternativas

A) Alternativa incorreta.

Justificativa: o diretório/media contém subdiretórios onde são montados dispositivos de mídias removíveis inseridas no sistema computacional. Por exemplo, um CD, inserido no drive de CD/DVD, é montado em um diretório criado dentro de /media, que nos permite acessar o seu conteúdo.

B) Alternativa incorreta.

Justificativa: o /dev é o diretório que contém os arquivos que representam os dispositivos de E/S do sistema. Por exemplo, um disco rígido do sistema estará representado como /dev/sda.

C) Alternativa incorreta.

Justificativa: o diretório /boot contém os arquivos utilizados para inicializar o sistema, como os arquivos do carregador de inicialização e do kernel do Linux.

D) Alternativa correta.

Justificativa: o diretório /bin possui os arquivos dos programas do sistema que são utilizados com frequência pelos usuários.

E) Alternativa incorreta.

Justificativa: o /etc é o diretório que contém os arquivos globais de configuração do sistema.

Questão 2. (FGV 2018, adaptada) Os SOs modernos do tipo Unix-like, como o Debian 9, oferecem aos administradores e aos analistas de suporte o mecanismo PAM (pluggable authentication modules). Nesse contexto, avalie as afirmativas.

I – O PAM é um mecanismo de segurança.

II – Os módulos do PAM podem especificar funções de mudança de senhas de usuários.

III – Os administradores não têm poder de configurar o comportamento de autenticação de cada serviço.

É correto apenas o que se afirma em:

A) I.

B) II.

C) III.

D) I e II.

E) I e III.

Resposta correta: alternativa D.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: PAM é um sistema de autenticação utilizado em SOs Unix-like, incluindo o Linux. Ele foi projetado para fornecer uma maneira de gerenciar a autenticação de usuários em diversos serviços e aplicativos no sistema, tornando mais fáceis a implementação de políticas de segurança personalizadas e a integração com diferentes métodos de autenticação. Logo, ele é considerado um mecanismo de segurança.

II – Afirmativa correta.

Justificativa: os módulos do PAM podem especificar métodos de autenticação, restrições de contas, funções de configuração de sessões e funções de mudança de senhas, de modo que, quando os usuários mudarem suas senhas, todos os mecanismos de autenticação necessários possam ser atualizados

imediatamente. Ou seja, os módulos PAM podem ser configurados para realizar várias tarefas relacionadas à autenticação, incluindo a mudança de senhas de usuários.

III – Afirmativa incorreta.

Justificativa: os administradores têm total poder de configurar o comportamento de autenticação de cada serviço ou aplicativo no sistema por meio dos módulos PAM, definindo, assim, políticas de autenticação específicas.

REFERÊNCIAS

Audiovisuais

REVOLUTION OS. 2012. 1 vídeo (85 min.). Publicado por Jardel Sacramento. Disponível em: https://shre.ink/n9PR. Acesso em: 25 set. 2023.

Textuais

ALVES, D. *Sistemas operacionais de redes (Windows/Linux)*. São Paulo: Sol, 2018.

ANDROID. *Arquitetura da plataforma*. 2020. Disponível em: https://shre.ink/nYeX. Acesso em: 25 set. 2023.

ANDROID. *Projeto de código aberto Android*. [s.d.]. Disponível em: https://shre.ink/nYe3. Acesso em: 25 set. 2023.

ANDROID. *Visão geral do gerenciamento de memória*. 2023. Disponível em: https://shre.ink/nYeo. Acesso em: 25 set. 2023.

ANVAARI, M.; JANSEN, S. *Evaluating architectural openness in mobile software platforms*. *In:* PROCEEDINGS OF THE FOURTH EUROPEAN CONFERENCE ON SOFTWARE ARCHITECTURE, 10., New York, 2010.

APPLE. *About Apple file system*. 2020. Disponível em: https://shre.ink/nOQm. Acesso em: 25 set. 2023.

APPLE. *Introdução à segurança da plataforma Apple*. 2022. Disponível em: https://shre.ink/nOQq. Acesso em: 25 set. 2023.

BALL, B.; DUFF, H. *Dominando Linux*: Red Hat e Fedora. São Paulo: Pearson, 2004.

BINNIE, C. *Segurança em servidores Linux*. São Paulo: Novatec, 2017.

BOSCH, J.; BOSCH-SIJTSEMA, P. From integration to composition: on the impact of software product lines, global development and ecosystems. *Journal of Systems and Software*, v. 83, p. 67-76, 2010.

CONHEÇA o iOS 8 para desenvolvedores. *DEVMEDIA*, 2015. Disponível em: https://shre.ink/nOIX. Acesso em: 25 set. 2023.

CÓRDOVA JUNIOR, R. S.; LEDUR, C. L.; MORAIS, I. S. *Sistemas operacionais*. Porto Alegre: Sagah, 2018.

DEITEL, H. M.; DEITEL, P. J.; CHOFFNES, D. R. *Sistemas operacionais*. São Paulo: Pearson Prentice-Hall, 2005.

DEITEL, H. M. *et al. Android*: como programar. 2. ed. Porto Alegre: Bookman, 2015.

ELAD, B. Linux statistics 2023 by market share, usage data and facts. *Enterprise Apps Today*, 2023. Disponível em: https://shre.ink/nO32. Acesso em: 25 set. 2023.

FERREIRA, R. *Linux*: guia do administrador do sistema. 2. ed. São Paulo: Novatec, 2003.

HAEDER, A. *et al. LPI Linux certification in a nutshell*: a desktop quick reference. 3. ed. Sebastopol: O'Reilly, 2010.

KROPIWIEC, D. D.; GEUS, P. L. *Paradigmas de segurança em sistemas operacionais*. Campinas: Universidade Estadual de Campinas, 2004. Disponível em: https://shre.ink/nO3t. Acesso em: 25 set. 2023.

LAUREANO, M. *Máquinas virtuais e emuladores*: conceitos, técnicas e aplicações. São Paulo: Novatec, 2006.

LEE, V.; SCHNEIDER, H.; SCHELL, R. *Aplicações móveis*: arquitetura, projeto e desenvolvimento. São Paulo: Pearson; Makron Books, 2005.

MACHADO, F. B.; MAIA, L. P. *Arquitetura de sistemas operacionais.* 5. ed. Rio de Janeiro: LTC, 2013.

MAGENTA, M.; NAKAMURA, M. *Learning Android.* 2. ed. Sebastopol: O'Reilly Media Inc., 2014.

MAZIERO, C. A. *Sistemas operacionais*: conceitos e mecanismos Curitiba: DINF; UFPR, 2019.

MAZZUCATO, M. *The entrepreneurial state*: Debunking public vs. private sector myths. London: Anthem, 2013.

MEIRELLES, F. S. *33ª pesquisa do uso de TI nas empresas*. São Paulo: FGV, 2022.

MENDONÇA, V. R. L.; BITTAR, T. J.; DIAS, M. S. *Um estudo dos sistemas operacionais Android e iOS para o desenvolvimento de aplicativos*. Catalão (GO): Enacomp, 2011.

MIRANDA, M. G. S. *Ecossistemas de software móveis*: um estudo exploratório sobre aspectos positivos e negativos no desenvolvimento de aplicações móveis. Belém: UFPA: 2016.

MOBILE operating system market share worldwide: Jan 2010 – Jan 2023. *Statcounter*, 2023a. Disponível em: https://shre.ink/nYiy. Acesso em: 25 set. 2023.

MOBILE operating system market share worldwide: Jan 2015 – Feb 2023. *Statcounter*, 2023b. Disponível em: https://shre.ink/nYtZ. Acesso em: 25 set. 2023.

MORAES, M. S. D. F. *et al. Fundamentos de desenvolvimento mobile*. Porto Alegre: Sagah, 2022.

NEGUS, C. *Linux, a Bíblia*: o mais abrangente e definitivo guia sobre Linux. Rio de Janeiro: Alta Books, 2014.

NEMETH, E.; SNYDER. G.; HEIN, T. R. *Manual completo do Linux*: guia do administrador. 2. ed. São Paulo: Pearson Books, 2007.

NIST. *An introduction to computer security*: the NIST handbook. Washington (EUA), 1995.

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

OLIVEIRA, W. P. *et al.* Desenvolvimento de simulações web para uma atividade de ensino-aprendizagem. *Revista Network Technologies*, v. 3, n. 1, 2009.

POPEK, G. J.; GOLDBERG, R. P. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, v. 17, n. 7, p. 412-421, 1974.

QUEIROZ, L. R. IPhone, Android, e a consolidação da cultura do smartphone: o papel do IPhone e do sistema operacional Android como catalisadores da consolidação no mercado de smartphones em escala global. *Revista Tecnologia e Sociedade*, v. 14, n. 30, p. 47-70, jan./abr. 2018. Disponível em: https://shre.ink/nOLM. Acesso em: 25 set. 2023.

ROCHA, A. M.; FINZI NETO, R. M. Introdução à arquitetura Apple iOS. *In:* ENCONTRO ANUAL DE COMPUTAÇÃO (ENACOMP), 9., Catalão (GO), Universidade Federal de Goiás, Catalão, 2011. Disponível em: https://shre.ink/nOMK. Acesso em: 25 set. 2023.

SADOWSKA, P. Android runtime – How Dalvik and ART work? *ProAndroidDev*, 2021. Disponível em: https://shre.ink/nYj0. Acesso em: 25 set. 2023.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Fundamentos de sistemas operacionais*. 9. ed. Rio de Janeiro: LTC: 2015.

SILVA, E. H. *Introdução à programação de computadores*: fabricação mecânica. Florianópolis: IF Santa Catarina, 2017. Disponível em: https://shre.ink/nOZH. Acesso em: 25 set. 2023.

SILVA, G. M. Linux: nível intermediário. Versão 6.02. *GuiaFoca*, 2020a. Disponível em: https:// shre.ink/nOZq. Acesso em: 25 set. 2023.

SILVA, M. B. F. *Cibersegurança*: uma visão panorâmica sobre a segurança da informação na internet. Rio de Janeiro: Freitas Bastos, 2023.

SILVA, M. B. F. *Segurança de sistemas operacionais (Windows/Linux)*. São Paulo: Sol, 2020b.

SNIA. *Common RAID disk data format specification*. SNIA – Storage Networking Industry Association, 2009. (Version 2.0, Revision 19)

STALLINGS, W. *Operating systems*: internals and design principles. 8. ed. Boston: Pearson, 2015.

TANENBAUM, A. S.; BOS, H. *Sistemas operacionais modernos*. 4. ed. São Paulo: Pearson: 2016.

TANENBAUM, A. S.; WOODHULL, S. *Sistemas operacionais*: projeto e implementação – o livro do Minix. 3. ed. Porto Alegre: Bookman, 2008.

YATES, M. Practical investigations of digital forensics tools for mobile devices. *In:* INFORMATION SECURITY CURRICULUM DEVELOPMENT CONFERENCE (InfoSecCD), 10., p. 156-162, New York, ACM, New York, 2010.

Informações:
www.sepi.unip.br ou 0800 010 9000