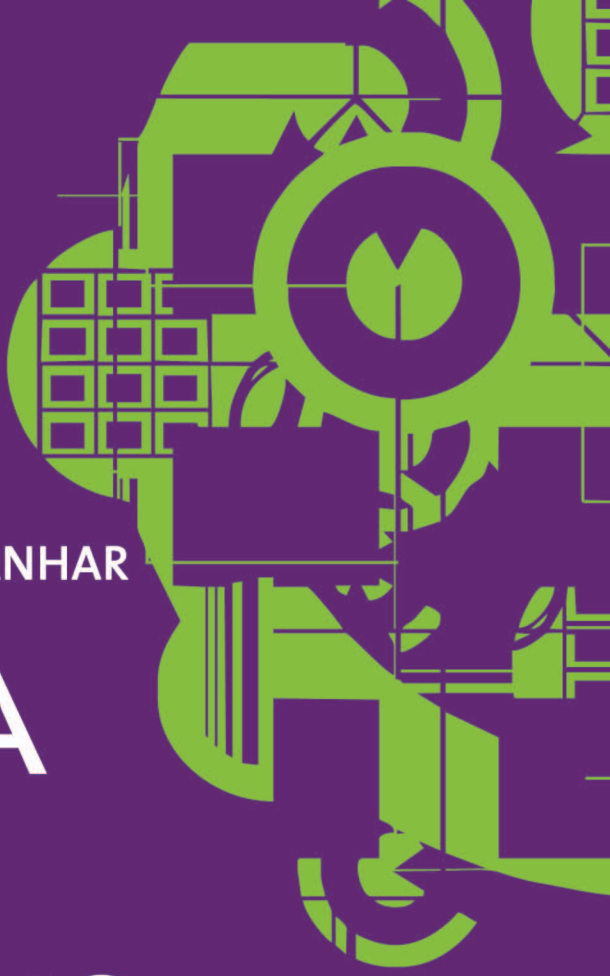
FRANCIS BERENGER MACHADO
LUIZ PAULO MAIA



MATERIAL SUPLEMENTAR PARA ACOMPANHAR

ARQUITETURA DE SISTEMAS OPERACIONAIS

INCLUINDO EXERCÍCIOS COM O SIMULADOR SOSIM E QUESTÕES DO ENADE

5ª EDIÇÃO

gen | LTC

Arquitetura de Sistemas Operacionais

5ª edição

Francis Berenger Machado (berenger@pobox.com)
Mestre em Administração de Empresas na PUC-Rio, graduado em Engenharia Eletrônica pelo CEFET-RJ e em Informática e Psicologia pela PUC-Rio. Atualmente é sócio da Descartes Consultoria (www.descartes-consultoria.com.br), além de lecionar na PUC-Rio e IBMEC. É autor do livro *Fundamentos de Sistemas Operacionais*.

Luiz Paulo Maia (lpmaia13@gmail.com)
Mestre em Informática pela UFRJ/NCE, graduado em Informática pela PUC-Rio e pós-graduado em Marketing pela mesma instituição. Atualmente é professor no Instituto Infnet e consultor na área de sistemas operacionais, redes e segurança. É autor dos livros *Arquitetura de Redes de Computadores* e *Fundamentos de Sistemas Operacionais*, e autor do software SOsim.

gen | LTC

Este Material Suplementar contém Estudos de Casos que podem ser usados como apoio para o livro **Arquitetura de Sistemas Operacionais**, 5ª edição, 2013. O acesso aos materiais suplementares desta edição está sujeito ao cadastramento no site da LTC — LIVROS TÉCNICOS E CIENTÍFICOS EDITORA LTDA.

LTC — Livros Técnicos e Científicos Editora Ltda.
Uma editora integrante do GEN | Grupo Editorial Nacional

Travessa do Ouvidor, 11
Rio de Janeiro, RJ — CEP 20040-040
Tels.: 21-3543-0770 / 11-5080-0770
Fax: 21-3543-0896
ltc@grupogen.com.br
www.ltceditora.com.br

Capa: Leônidas Leite

Editoração Eletrônica: Anthares

Sumário

Estudos de Casos

Microsoft Windows

1. Histórico

A Microsoft lançou em 1981 seu primeiro sistema operacional, o MS-DOS (Disk Operating System), para a linha de computadores pessoais IBM-PC, concebido para ser um sistema operacional de 16 bits, monoprogramável, monousuário e com uma interface de linha de comando. O MS-DOS foi desenvolvido com base no sistema operacional CP/M e algumas ideias do Unix.

Em 1985, é lançada a primeira versão do MS Windows, que introduziu uma interface gráfica, porém manteve o MS-DOS como o sistema operacional. As versões posteriores do MS Windows, como Windows 3.0, Windows 95, Windows 98 e Windows Me, apesar de várias melhorias e inovações, sempre tinham o MS-DOS como o núcleo do sistema operacional.

Devido às inúmeras limitações e deficiências do MS-DOS, a Microsoft começou a conceber um novo sistema operacional em 1988, conhecido como Windows NT (New Technology). Este novo projeto foi conduzido por David Cutler, ex-projetista da Digital Equipment Corporation (DEC), que foi responsável pelo desenvovimento de inúmeros sistemas operacionais, como o PDP/RSX e o VAX/VMS. Além da grande influência do sistema operacional VMS, no projeto do Windows NT foram utilizados vários conceitos dos sistemas OS/2 e LAN Manager.

Em 1993, a Microsoft lança o Windows NT nas versões para desktop e servidores, sistema operacional de 32 bits, com multitarefa preemptiva, multithread, memória virtual e suporte a múltiplos processadores simétricos. O Windows NT não tem qualquer relação com a arquitetura do MS-DOS, mas oferece compatibilidade parcial com aplicações legadas, além de ter incorporado algumas de suas características, como a interface gráfica. Com isso, passaram a existir duas linhas de sistemas operacionais com arquiteturas completamente distintas, porém com a mesma interface para o usuário (Fig. 1).

O Windows 2000 é uma evolução do Windows NT versão 4, pois mantém a mesma arquitetura interna. O Windows 2000 passou a incorporar alguns recursos da família DOS-Windows, como a função de plug-and-play. A grande novidade trazida pelo sistema é o Active Directory, que funciona como um serviço de diretórios e veio substituir o conceito de domínio existente no Windows NT.

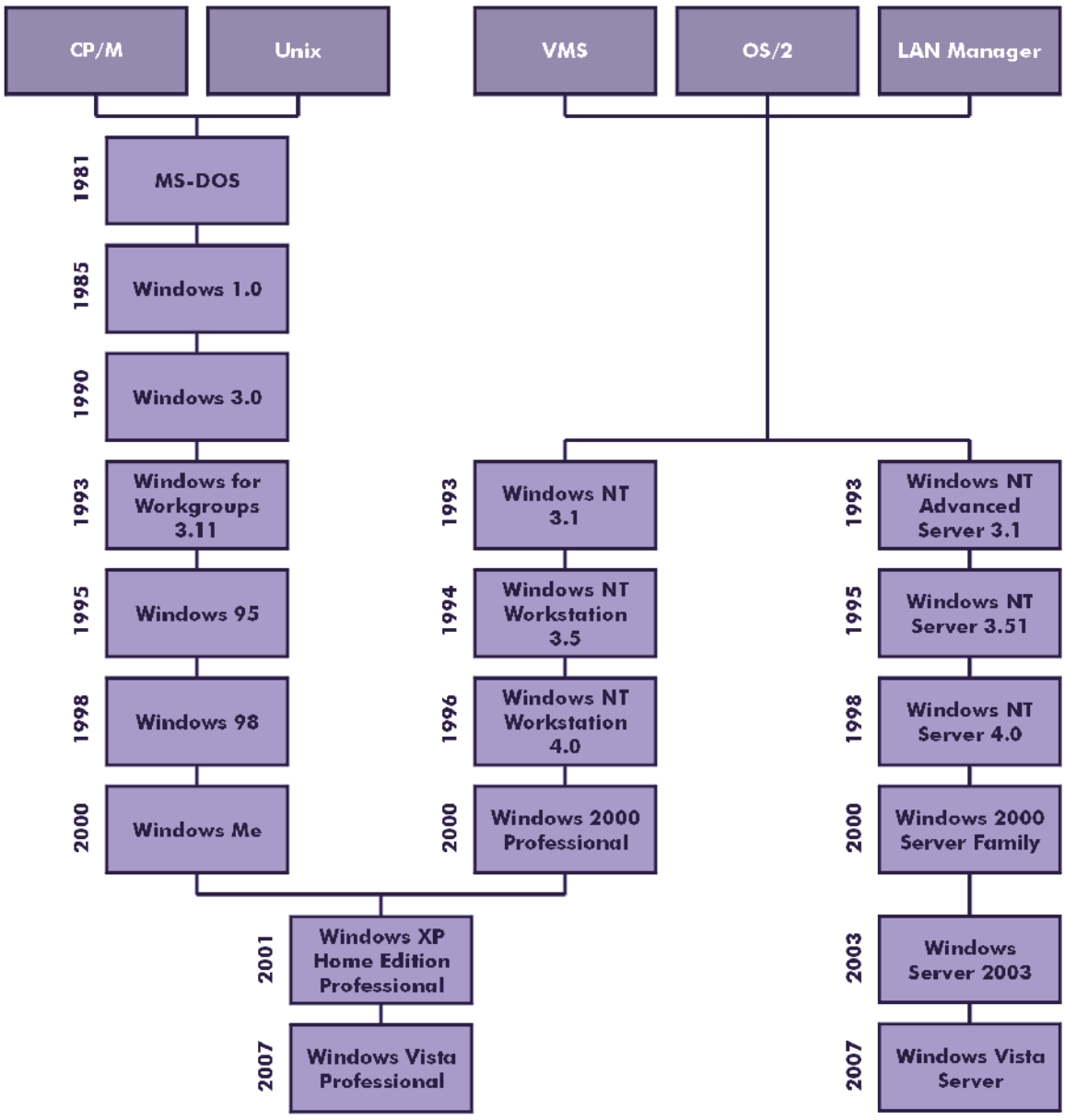


Fig. 1 Histórico dos sistemas operacionais MS Windows.

O Windows XP, lançado em 2001, introduz uma nova interface gráfica e alguns poucos recursos adicionais, porém mantém a mesma arquitetura do Windows 2000. A partir do Windows XP, a Microsoft começou a descontinuar as famílias DOS-Windows e Windows NT/2000, integrando as duas linhas de sistemas operacionais. Em 2003, é lançado o Windows Server 2003, nova versão do Windows 2000 Server, com suporte a processadores de 64 bits.

Em 2007/2008, a Microsoft lança o Windows Vista para desktops e servidores. Sua expectativa é que esta nova versão do MS Windows seja o sistema operacional utilizado em grande escala na década de 2010.

2. Características

Este capítulo apresenta as características do Windows Server 2003, sistema operacional multiprogramável de 32/64 bits, que suporta escalonamento preemptivo, multithread, multiusuário, multiprocessamento simétrico (SMP) e gerência de memória virtual.

O Windows Server 2003 apresenta seis versões, que possuem a mesma arquitetura interna e interface gráfica. A diferença entre as versões está no suporte ao número de processadores, tamanho da memória física, número de conexões de rede e serviços oferecidos. A Tabela 1 apresenta uma comparação entre as principais características das diferentes versões do Windows Server 2003.

Tabela 1 Versões do Windows Server 2003

Versão	Máximo UCPs (32 bits)	Máximo Memória (32 bits)	Máximo UCPs (64 bits)	Máximo Memória (64 bits)
Windows Server 2003 Web Edition	2	2 GB	–	–
Windows Server 2003 Small Business Server	2	2 GB	–	–
Windows Server 2003 Standard Edition	4	4 GB	–	–
Windows Server 2003 Enterprise Edition	8	32 GB	8	64 GB
Windows Server 2003 Datacenter Edition	32	128 GB	64	1024 GB

3. Estrutura do Sistema

O MS Windows possui mais de 40 milhões de linhas de código escritas, em sua maioria, em Linguagem C, porém com alguns módulos desenvolvidos em C++ e assembly. O sistema é estruturado combinando o modelo de camadas e o modelo cliente-servidor. Embora não seja totalmente orientado a objetos, o MS Windows representa seus recursos internos como objetos. Essa abordagem diminui o impacto das mudanças que o sistema venha a sofrer no futuro, reduzindo o tempo e o custo de manutenção. Além disso, criação, manipulação, proteção e compartilhamento de recursos podem ser feitos de forma simples e uniforme. Quando um objeto é criado, um handle é associado a ele, permitindo o seu acesso e compartilhamento. A Tabela 2 apresenta alguns objetos implementados pelo sistema.

Tabela 2 Objetos do Windows 2003

Objetivo	Descrição
Processo	Ambiente para execução de um programa.
Thread	Unidade de execução em um processo.
Seção	Área compartilhada de memória.
Porta	Mecanismo para a troca de mensagens entre processos.
Token de acesso	Identificação de um objeto.
Evento	Mecanismo de sincronização entre processos.
Semáforo	Contador utilizado na sincronização entre processos.
Timer	Temporizador.

A arquitetura do MS Windows pode ser dividida em duas camadas, como pode ser observado na Fig. 2. No modo usuário estão os processos do sistema, serviços, aplicações, subsistemas de ambiente e subsistemas de DLLs. No modo kernel está o núcleo do sistema propriamente dito e o HAL.

Hardware Abstraction Layer

O Hardware Abstraction Layer (HAL) é uma biblioteca que engloba a parte do código do sistema dependente do hardware, como acesso a registradores e endereçamento de dispositivos, identificação de interrupções, DMA, temporização, sincronização em ambientes com múltiplos processadores e interface com a BIOS e CMOS. Essa camada garante ao Windows uma facilidade muito grande de ser portado para diferentes plataformas de hardware.

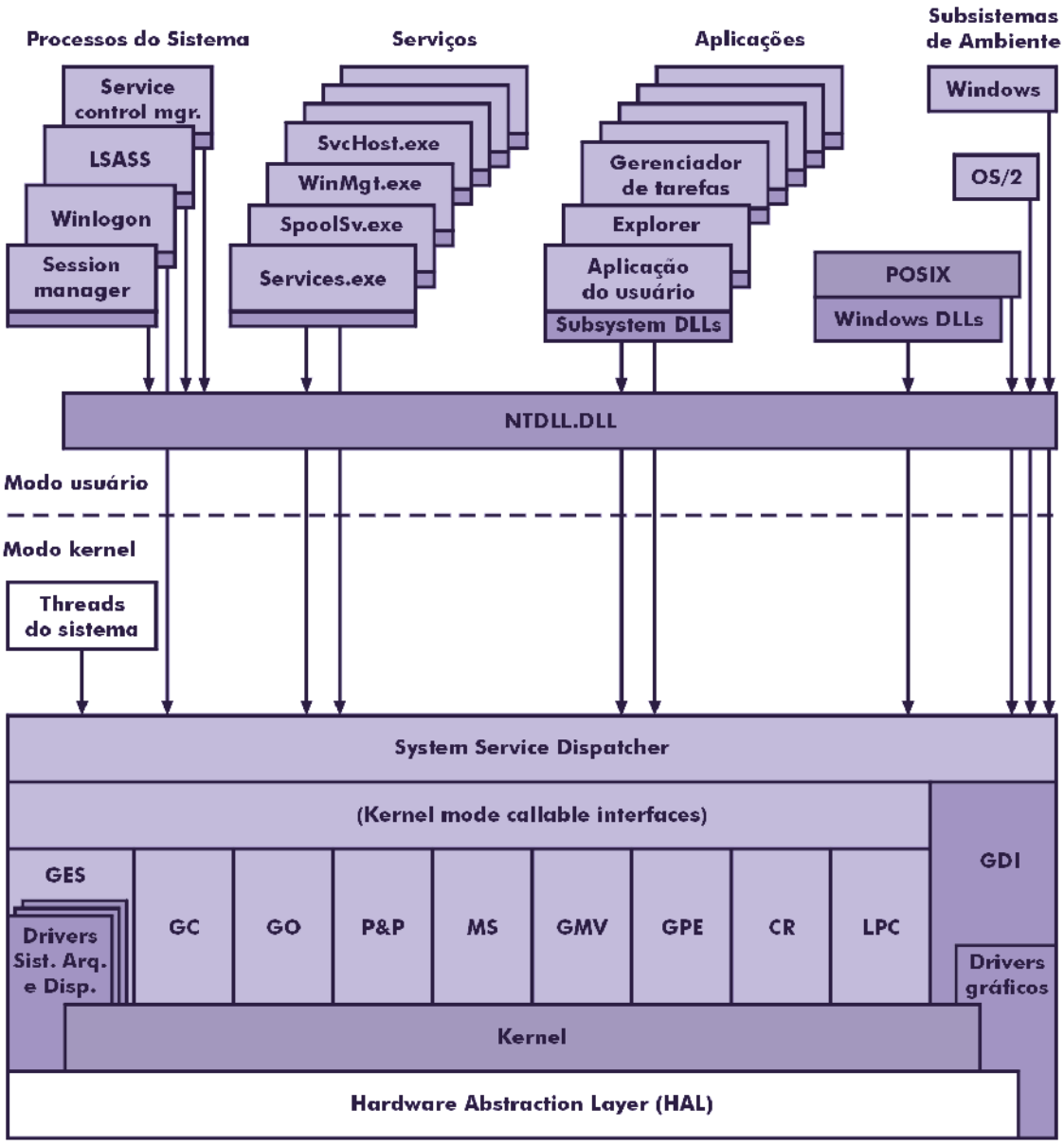


Fig. 2 Arquitetura do sistema.

- **Kernel**

  O kernel é responsável por tornar o sistema operacional totalmente independente do hardware, implementado as demais funções não suportadas pelo HAL, como troca de contexto, escalonamento e dispatching, tratamento de interrupções e exceções, e suporte a objetos de uso interno do kernel, como DPC (Deferred Procedure Call), APC (Asynchronous Procedure Call) e dispatcher objects.

- **Device Drivers**

  No MS Windows, um device driver não manipula diretamente um dispositivo. Esta tarefa é função do HAL, que serve de interface entre o driver e o hardware. Esse modelo permite que a maioria dos device drivers seja escrita em Linguagem C e, consequentemente, tenha grande portabilidade.

- **Executivo**

  O executivo é a parte do núcleo totalmente independente do hardware. Suas rotinas podem ser agrupadas em categorias, em função do tipo de serviço que oferecem, como gerência de objetos (GO), gerência de E/S (GES), gerência de processos e threads (GPE), gerência de memória virtual (GMV), monitor de segurança (MS), gerência de cache (GC), gerenciamento de energia e plug-and-play (P&P), configuração do registry (CR), Local Procedure Call (LPC) e Graphics Device Interface (GDI).

- **NTDLL.DLL**

  O NTDLL é uma biblioteca do sistema que faz a interface entre as aplicações que utilizam o subsistema de DLLs e o executivo. Uma DLL (Dynamic Link Libraries) é uma biblioteca de procedimentos, geralmente relacionados, que são linkados à aplicação apenas em tempo de execução. Esse tipo de biblioteca compartilhada evita que aplicações que utilizem os mesmos procedimentos tenham sua própria cópia individual das rotinas na memória principal.

- **Subsistemas**

  O MS Windows oferece três subsistemas de ambiente: Windows, OS/2 e POSIX, que permitem que aplicações desenvolvidas nestes ambientes sejam suportadas pelo mesmo sistema operacional. Enquanto os subsistemas OS/2 e POSIX são opcionais, o subsistema Windows é obrigatório e deve estar sempre carregado.

  As aplicações que realizam chamadas às rotinas do sistema utilizam uma biblioteca, conhecida como Application Program Interface (API). Uma aplicação no MS Windows não pode realizar uma chamada diretamente a uma rotina do sistema utilizando uma system call, mas sempre fazendo uso de uma API. As APIs funcionam como uma interface entre a aplicação e as system calls propriamente ditas, que por sua vez fazem as chamadas às rotinas do sistema operacional. As APIs são implementadas utilizando o subsistema de DLLs.

4. Processos e Threads

Processos no MS Windows são objetos criados e eliminados pelo gerente de objetos, representados internamente por uma estrutura chamada EPROCESS (Executive Process Block). Um processo ao ser criado possui, dentre outros recursos do sistema, um espaço de endereçamento virtual, uma tabela de objetos, um token de acesso e pelo menos um thread (Fig. 3).

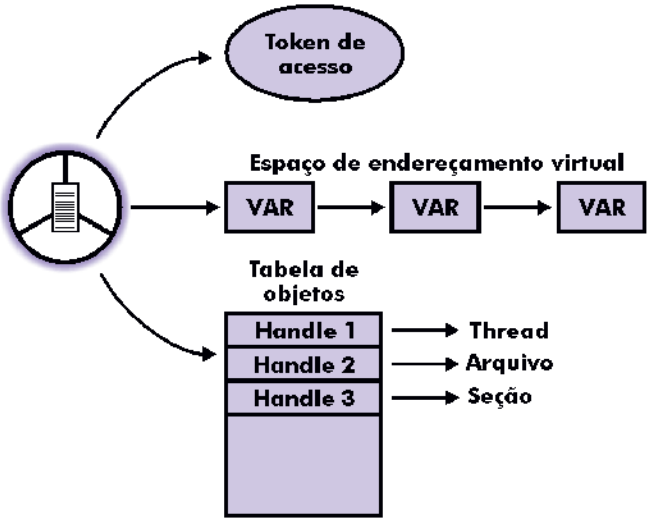


Fig. 3 Estrutura do processo.

O espaço de endereçamento virtual é formado por descritores, conhecidos como VARs (Virtual Address Descriptors), que permitem que a alocação do espaço de endereçamento seja feita dinamicamente, conforme as referências aos endereços virtuais. A tabela de objetos contém handles para cada objeto a que o processo tem acesso. Um handle é criado quando o processo cria um novo objeto ou solicita acesso a um já existente. O token de acesso identifica o processo para o sistema operacional, e sempre que o processo tem acesso a um recurso do sistema o token é utilizado para determinar se o acesso é permitido. Cada processo é criado com um único thread, mas threads adicionais podem ser criados e eliminados quando necessário.

Threads também são implementados como objetos, sendo criados e eliminados pelo gerenciador de objetos e representados por uma estrutura chamada ETHREAD (Executive Thread Block). Todos os threads de um processo compartilham o mesmo espaço de endereçamento virtual e todos os recursos do processo, incluindo token de acesso, prioridade-base, tabela de objetos e seus handles.

O MS Windows escalona apenas threads para execução, e não processos. A Tabela 3 apresenta os diferentes estados possíveis de um thread, enquanto a Fig. 4 apresenta as mudanças de estado de um thread durante sua existência.

Tabela 3 Estados de um thread

Estado	Descrição
Criação	Criação e inicialização do thread.
Espera	O thread aguarda por algum evento, como uma operação de E/S.
Execução	O thread está sendo executado.
Pronto	O thread aguarda para ser executado.
Standby	O thread foi escalonado e aguarda pela troca de contexto para ser executado. Somente um thread pode estar no estado de standby para cada processador existente.
Terminado	Quando um thread termina sua execução, o sistema o coloca no estado de terminado, porém o objeto pode ou não ser eliminado.
Transição	O thread aguarda que suas páginas gravadas em disco sejam lidas para a memória principal.

Além de processos e threads, o MS Windows implementa os conceitos de job e fiber. Um job é uma coleção de processos que guardam alguma relação e devem ser gerenciados como uma unidade. Processos em um job compartilham quotas e privilégios, como o número máximo de processos que podem ser criados e o espaço máximo alocado na memória principal.

O MS Windows implementa threads em modos usuário e kernel. Threads em modo kernel, denominados simplesmente threads, apresentam problemas de desempenho devido à necessidade de troca de modo de acesso usuário-kernel-usuário. Threads em modo usuário, denominados fibers, eliminam as trocas de contexto e de modo de acesso e, consequentemente, oferecem melhor desempenho. Cada thread pode ter múltiplos fibers, da mesma forma que um processo pode ter múltiplos threads. O sistema operacional desconhece a existência dos fibers, ficando a cargo da própria aplicação o seu escalonamento.

A Tabela 4 apresenta algumas APIs utilizadas para criação, eliminação, comunicação e sincronização de processos, threads e fibers.

5. Gerência do Processador

O MS Windows suporta dois tipos de política de escalonamento: escalonamento circular com prioridades e escalonamento por prioridades. A política de escalonamento é implementada associando prioridades aos processos e threads. Inicialmente, o thread recebe a prioridade do processo

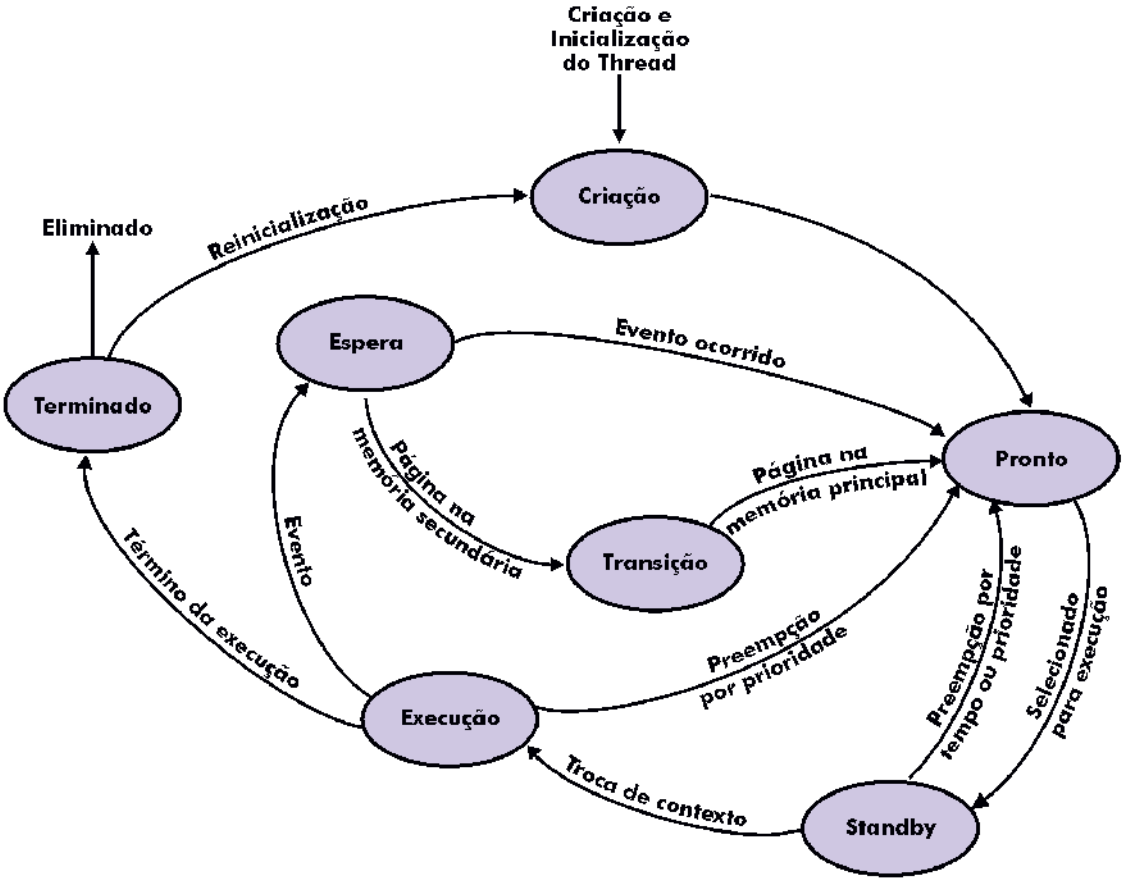


Fig. 4 Mudanças de estado de um thread.

Tabela 4 As APIs para a gerência de processos, threads e fibers

API	Descrição
CreateProcess	Cria um processo.
CreateThread	Cria um thread no processo corrente.
CreateRemoteThread	Cria um thread em um outro processo.
CreateFiber	Cria um fiber no processo corrente.
OpenProcess	Retorna um handle para um determinado processo.
GetCurrentProcessID	Retorna a identificação do processo corrente.
ExitProcess	Finaliza o processo corrente e todos seus threads.
TerminateProcess	Termina um processo.
ExitThread	Finaliza o thread corrente.
TerminateThread	Termina um thread.
CreateSemaphore	Cria um semáforo.
OpenSemaphore	Retorna um handle para um determinado semáforo.
WaitForSingleObject	Espera que um único objeto, como um semáforo, seja sinalizado.
WaitForMultipleObjects	Espera que um conjunto de objetos sejam sinalizados.
EnterCriticalSection	Sinaliza que a região crítica está sendo executada.
LeaveCriticalSection	Sinaliza que a região crítica não está mais sendo executada.

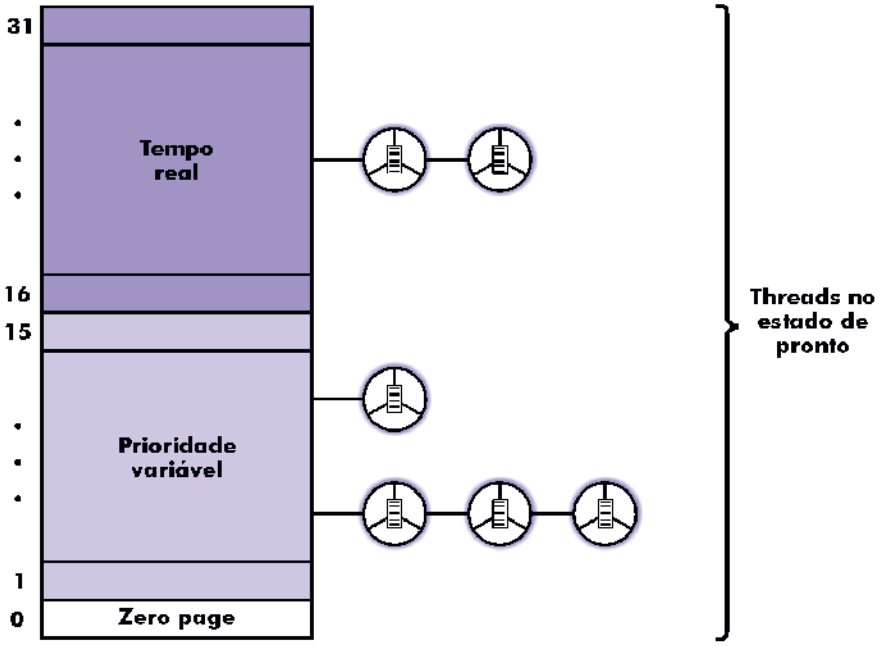


Fig. 5 Níveis de prioridade.

ao qual pertence, podendo ser alterada posteriormente. O escalonamento é realizado considerando apenas os threads no estado de pronto, sendo que os processos servem apenas como unidade de alocação de recursos.

O MS Windows implementa 32 níveis de prioridades, divididos em duas faixas: escalonamento de prioridade variável (1-15) e de tempo real (16-31). A prioridade zero serve apenas para que um thread especial do sistema (zero page) seja executado quando não existirem threads prontos para execução. Threads com a mesma prioridade são organizados em filas no esquema FIFO, e o primeiro thread na fila de maior prioridade será sempre selecionado para execução. Caso um thread fique pronto com prioridade maior que o thread em execução, ocorrerá a preempção por prioridade (Fig. 5).

A seguir, são apresentados os dois esquemas de escalonamento implementado pelo Windows:

- **Prioridade Variável**

  No esquema de prioridade variável (1-15) é adotada a política de escalonamento circular com prioridades. O thread de maior prioridade é selecionado e recebe uma fatia de tempo para ser executado. Terminado este *quantum*, o thread sofre preempção por tempo e retorna para o estado de pronto no final da fila associada a sua prioridade.

  O escalonamento de prioridade variável utiliza dois tipos de prioridades para cada thread: base e dinâmica. A prioridade-base é normalmente igual ao do processo e não sofre alteração durante a existência do thread. Por outro lado, a prioridade dinâmica é sempre igual ou maior que a prioridade-base e varia de acordo com as características do thread, sendo controlada pelo sistema. O escalonamento é realizado em função da prioridade dinâmica dos threads.

  A prioridade dinâmica é aplicada exclusivamente aos threads que passam pelo estado de espera. Por exemplo, sempre que um thread realiza uma operação de E/S o sistema o coloca no estado de espera. No momento em que o thread é transferido para o estado de pronto, a prioridade-base do thread é acrescida de um valor em função do tipo de operação realizado, ou seja, a prioridade dinâmica é resultado da soma da prioridade-base mais um incremento

(Tabela 5). O incremento não é cumulativo, e a prioridade dinâmica nunca poderá ultrapassar o limite máximo da prioridade variável, ou seja, prioridade 15.

Tabela 5 Incremento na prioridade-base

Tipo de Operação de E/S	Incremento
Disco, CD e vídeo	1
Rede	2
Teclado e mouse	6
Som	8

O mecanismo de prioridade dinâmica permite equilibrar o uso do processador entre os diversos threads no sistema. Por exemplo, suponha que existam apenas dois threads no sistema com a mesma prioridade, um CPU-bound e outro I/O-bound. Toda vez que o thread I/O-bound realizar uma operação de E/S e retornar ao estado de pronto, o sistema incrementará a sua prioridade-base, fazendo com que o thread CPU-bound sofra uma preempção. Dessa forma, enquanto o thread I/O-bound executa várias vezes, utilizando apenas parte da sua fatia de tempo, o thread CPU-bound executa menos vezes, porém utilizando todo o seu *quantum*.

- **Tempo Real**

  No esquema de tempo real (16-31) é adotada a política de escalonamento por prioridades. Existem duas grandes diferenças no esquema de tempo real, se comparado ao de prioridade variável: o conceito de fatia de tempo e a prioridade dinâmica.

  Os threads no estado de execução são processados o tempo que for necessário, não existindo fatia de tempo. Nesse caso, um thread é interrompido apenas por processos mais prioritários, ou seja, quando sofre uma preempção por prioridade. Além disso, os threads não têm aumentos de prioridade, sendo as prioridades-base e dinâmicas sempre iguais.

  Esta faixa de prioridade deve ser utilizada apenas por threads do sistema operacional por questões de segurança. Se qualquer thread com prioridade acima de 15 entrar em loop, nenhum dos threads com prioridades entre 1 e 15 será executado.

Os dois níveis de escalonamento apresentados permitem ao MS Windows oferecer características de sistemas de tempo compartilhado e tempo real dentro do mesmo ambiente, tornado-o versátil e possibilitando sua utilização por diferentes tipos de aplicações. A Tabela 6 apresenta algumas APIs relacionadas ao escalonamento de threads.

Tabela 6 APIs de escalonamento

API	Descrição
SuspendThread	Coloca o thread no estado de espera.
ResumeThread	Permite que o thread volte para o estado de pronto.
SetPriorityClass	Permite alterar a prioridade-base do processo.
SetThreadPriority	Permite alterar a prioridade-base do thread.
SetThreadPriorityBoost	Permite oferecer um incremento de prioridade temporário ao thread.
Sleep	Coloca o thread em estado de espera por um certo intervalo de tempo.

6. Gerência de Memória

O MS Windows implementa o mecanismo de gerência de memória virtual por paginação. A gerência de memória, ao contrário do escalonamento, lida com processos e não threads, ou seja, o espaço de endereçamento virtual pertence ao processo. O tamanho do espaço de endereçamento virtual

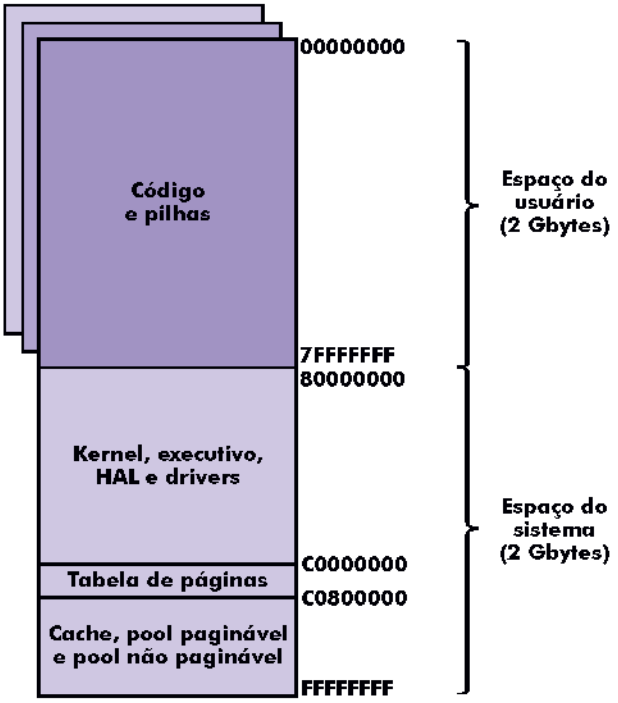


Fig. 6 Espaço de endereçamento virtual em uma arquitetura de 32 bits.

depende da arquitetura de hardware. Em máquinas com 32 bits, cada processo pode endereçar 4 Gb, sendo 2 Gb reservados para o sistema operacional e 2 Gb para aplicações do usuário. Cada processo tem o seu espaço de endereçamento virtual único, independente dos demais processos, enquanto o espaço de endereçamento do sistema é compartilhado por todos os processos. A Fig. 6 apresenta o espaço de endereçamento virtual em uma arquitetura de 32 bits.

No MS Windows, o tamanho da página é definido em função da arquitetura do processador, possibilitando páginas entre 4 Kb (32 bits) e 2 Mb (64 bits). No processador Pentium, da Intel, as páginas possuem 4 Kb. Em arquiteturas de 32 bits, o mapeamento é realizado utilizando tabelas de páginas em dois níveis, sendo que a tabela de primeiro nível (page directory index) aponta para tabelas de segundo nível (page table index) que, por sua vez, apontam para os frames na memória principal (Fig. 7). O endereço real é obtido combinando-se o endereço do frame e o deslocamento. Em arquiteturas de 64 bits, o MS Windows utiliza um esquema de mapeamento em quatro níveis, ou seja, utilizando quatro tabelas de endereçamento.

Cada página da memória virtual gera uma entrada na tabela de páginas denominada *page table entry* (PTE). O PTE possui, entre outras informações, bits que indicam a localização da página, um bit que indica se a página foi alterada (*dirty bit*) e bits que permitem implementar a proteção da página.

A gerência de memória permite que dois ou mais processos compartilhem a mesma área de código ou dados na memória principal. Não existe nenhum mecanismo automático de controle de acesso a essa região, ficando como responsabilidade de cada processo a sincronização para evitar conflitos. Além disso, é possível mapear um arquivo residente em disco na memória principal e compartilhá-lo entre diversos processos. As aplicações podem ler e gravar registros diretamente na memória principal, evitando operações de E/S em disco.

O MS Windows implementa o esquema de paginação por demanda como política de busca de páginas. Quando ocorre um *page fault*, o sistema carrega para a memória principal a página referenciada e um pequeno conjunto de páginas próximas (cluster de páginas), na tentativa de

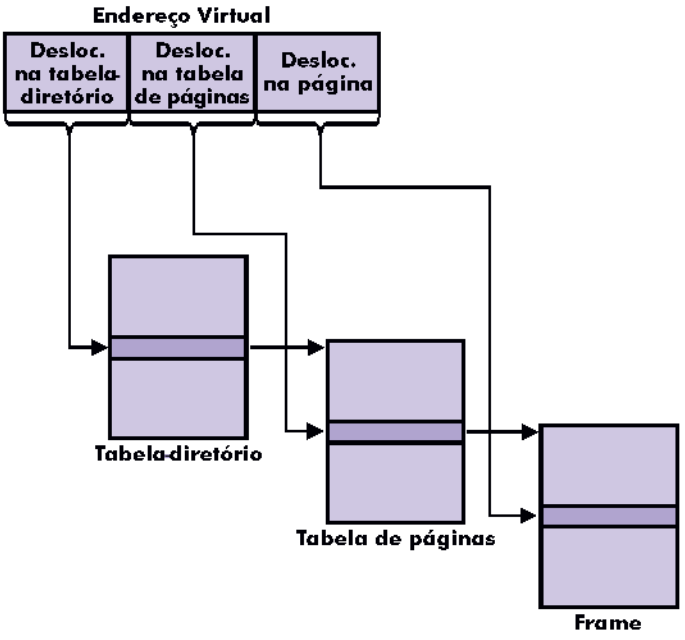


Fig. 7 Mapeamento.

reduzir o número de operações de leitura em disco e melhorar o desempenho do sistema (Fig. 8a). O tamanho do cluster de páginas varia conforme o tamanho da memória principal e se a página lida for de código ou dados.

Quando um programa é executado, apenas parte de seu código e dados estão na memória principal. O conjunto de frames que um processo possui na memória principal é denominado *working set*. As páginas pertencentes ao processo são mantidas em uma estrutura chamada lista do working set, organizada no esquema FIFO. O tamanho do working set varia conforme a taxa de paginação do processo e da disponibilidade de frames na memória principal. Os tamanhos mínimo e máximo do working set são definidos na criação do processo, mas podem ser alterados durante a sua existência. É importante ressaltar que o conceito de working set no MS Windows não é idêntico ao apresentado no Cap. 10 — Gerência de Memória Virtual.

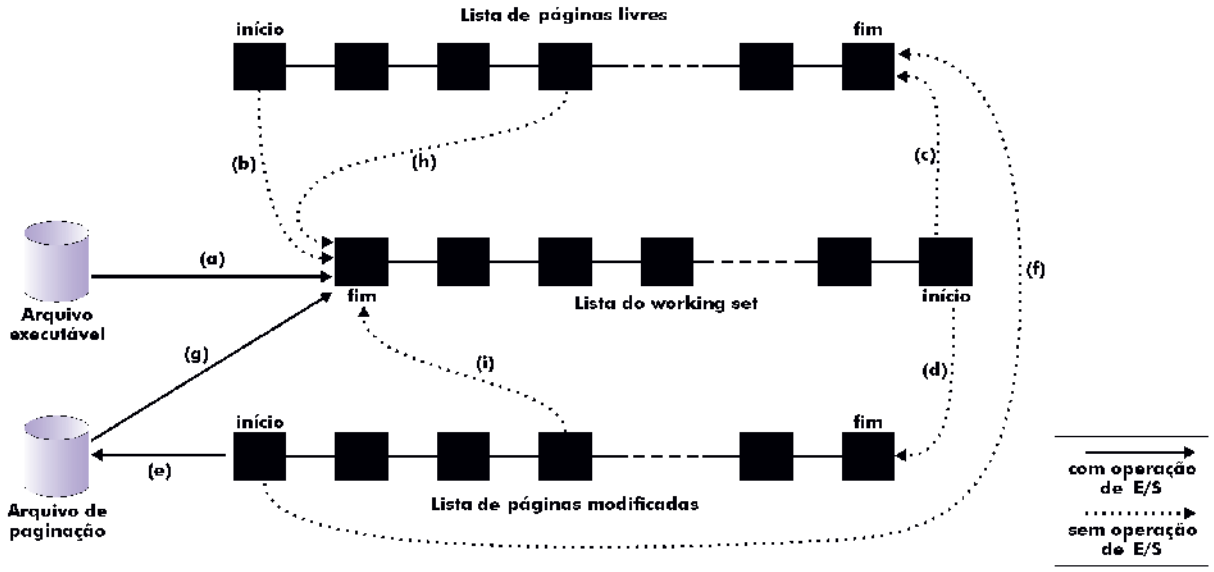


Fig. 8 Gerência de memória.

O MS Windows implementa o esquema FIFO com buffer de páginas como política de substituição de páginas. De forma simplificada, o sistema possui duas listas que funcionam como buffers: a lista de páginas livres (*free page list*) e a lista de páginas modificadas (*modified page list*). As listas de páginas têm a função de reduzir a taxa de page faults dos processos e, principalmente, o número de operações de leitura e gravação em disco.

A lista de páginas livres (LPL) agrupa todos os frames disponíveis para uso na memória principal. Quando um processo necessita de uma nova página e seu working set não atingiu o seu limite máximo, o sistema retira a primeira página da LPL e adiciona ao final da lista do working set do processo (Fig. 8b). Por outro lado, quando o working set do processo alcança seu limite máximo, o processo deve ceder a página mais antiga no seu working set para a LPL, antes de receber uma nova página (Fig. 8c). Como pode ser observado, a política de substituição de páginas é local, afetando apenas o processo que gerou o page fault.

O destino da página selecionada para substituição vai depender se a página sofre modificação. As páginas não modificadas não precisam de tratamento especial, pois podem ser recuperadas do arquivo executável quando necessário (Fig. 8a). Já uma página modificada não pode ser simplesmente descartada quando sai do working set do processo, pois contém dados necessários à continuidade da execução do programa. Nesse caso, a página selecionada não é colocada no final da LPL, mas na lista de páginas modificadas (LPM). A LPM agrupa páginas de todos os processos no sistema e tem a função de adiar a gravação em disco das páginas modificadas (Fig. 8d).

A lista de páginas livres, eventualmente, pode ficar com poucas páginas disponíveis para atender à demanda dos processos. Nesse caso, as páginas que estão na LPM são gravadas em disco no arquivo de paginação (Fig. 8e) e transferidas para a LPL para aumentar o número de frames livres (Fig. 8f). O arquivo de paginação é uma área em disco comum a todos os processos, onde as páginas modificadas são salvas para quando forem posteriormente referenciadas (Fig. 8g). As páginas modificadas são gravadas no disco em conjunto e, portanto, apenas uma operação de gravação é efetuada, tornando o processo mais eficiente. O Windows permite múltiplos arquivos de paginação que podem ser distribuídos por vários discos, permitindo um desempenho ainda melhor das operações de leitura e gravação.

A LPL, além de agrupar os frames disponíveis da memória principal, permite também reduzir o impacto do algoritmo FIFO utilizado na política de substituição de páginas. Quando o sistema libera uma página do working set de um processo para a LPL, esta página não é imediatamente utilizada, pois o frame deverá percorrer a lista até chegar ao seu início. Desta forma, as páginas que saem do working set permanecem na memória por algum tempo, permitindo que essas mesmas páginas possam retornar ao working set do processo. Nesse caso, na ocorrência de um page fault o sistema poderá encontrar o frame requerido na LPL, gerando um page fault sem a necessidade de uma operação de leitura em disco (Fig. 8h). Por outro lado, quando o frame não é encontrado na LPL, o sistema é obrigado a realizar uma operação de leitura em disco para obter novamente a página referenciada (Fig. 8a). De forma semelhante, quando uma página modificada é referenciada o frame pode estar na LPM, dispensando uma operação de leitura em disco (Fig. 8i). Caso contrário, o frame deverá ser lido do arquivo de paginação em disco (Fig. 8g).

Periodicamente, o sistema verifica o tamanho da LPL e, caso o número de páginas esteja abaixo de um limite mínimo, o tamanho do working set dos processos é reduzido de forma a liberar páginas para a LPL (*working set trimming*). Inicialmente, o sistema seleciona os processos com grandes working sets e que estão inativos por mais tempo ou que apresentem baixas taxas de paginação.

Além da LPL e LPM, o sistema mantém outras três listas para a gerência de memória: lista de páginas ruins, lista de páginas zeradas e lista de páginas em espera. Para controlar todas as páginas e listas na memória principal, a gerência de memória mantém uma base de dados dos frames na memória (*page frame database*), com informações sobre todas as páginas livres e utilizadas.

7. Sistema de Arquivos

O MS Windows suporta quatro tipos de sistemas de arquivos: CDFS, UDF, FAT e NTFS. Cada sistema determina como os arquivos e diretórios são organizados, o formato dos nomes dos arquivos, desempenho e segurança de acesso aos dados. O CDFS (CD-ROM File System) oferece suporte a dispositivos como CD-ROMs e DVDs. O UDF (Universal Disk Format) é uma evolução do CDFS, e também é voltado para CDs e DVDs.

O sistema de arquivos FAT (*File Allocation Table*) foi desenvolvido para o sistema MS-DOS e, posteriormente, utilizado nas várias versões do MS Windows. O FAT16 utiliza o esquema de listas encadeadas para estruturar o sistema de arquivos, está limitado a partições de no máxino 4 Gb e apresenta baixo desempenho e segurança. O FAT32 possui a maioria das limitações do sistema de arquivo FAT, porém permite partições de até 8 Tb.

O NTFS (NT File System) foi desenvolvido especialmente para as novas versões do MS Windows, e utiliza o esquema de árvore-B para estruturar o sistema de arquivos, oferecendo alto grau de segurança e desempenho, além de inúmeras vantagens quando comparado aos sistemas FAT, como:

- nomes de arquivos com até 255 caracteres, incluindo brancos e letras maiúsculas e minúsculas;
- partições NTFS dispensam o uso de ferramentas de recuperação de erros;
- proteção de arquivos e diretórios por grupos e ACLs;
- criptografia e compressão de arquivos;
- suporte a volumes de até 16 Exabytes e 2^{32}-1 arquivos por volume;
- ferramentas de desfragmentação e gerência de quotas em disco;
- suporte a Unicode;
- suporte a RAID 0, RAID 1 e RAID 5.

O NTFS trabalha com volumes que são partições lógicas de um disco físico. Um volume pode representar todo o espaço de um disco ou apenas parte do disco físico. Além disso, em um mesmo disco podem ser configurados vários volumes com diferentes sistemas de arquivos, como NTFS e FAT. Esse esquema de particionamento permite o boot de diferentes tipos e versões de sistemas operacionais a partir de um único disco físico.

Um disco, quando formatado, é dividido em setores que são agrupados em clusters. O cluster é uma unidade de alocação de espaço em disco e seu tamanho varia em função do volume, ou seja, quanto maior o volume, maior o tamanho do cluster. O tamanho do cluster influencia na fragmentação interna do volume. Na Fig. 9, um volume NTFS pode ser visualizado como uma sequência de clusters de quatro setores, identificados por um Logical Cluster Number (LCN).

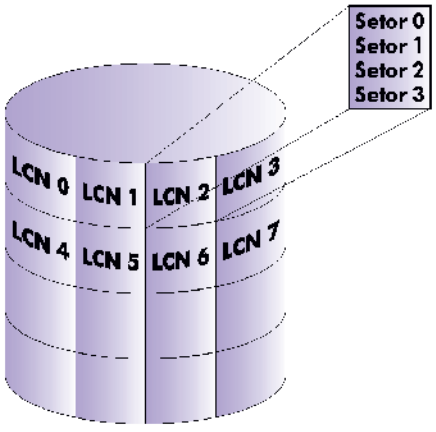


Fig. 9 Estrutura lógica do disco.

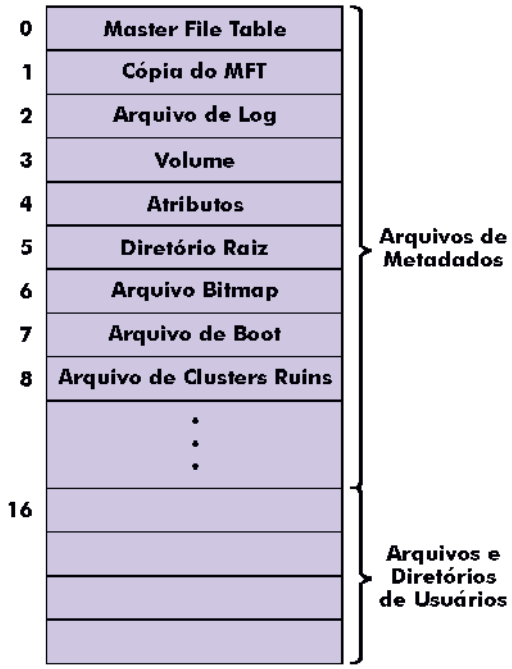


Fig. 10 Master File Table.

A estrutura de um volume NTFS é implementada a partir de um arquivo chamado Master File Table (MFT), formado por diversos registros de 1 Kb. A Fig. 10 apresenta a estrutura do MFT, onde os registros de 0 a 15 são utilizados para mapear os arquivos de controle do sistema de arquivos (arquivos de metadados) e os demais registros são utilizados para mapear arquivos e diretórios de usuários. O registro 0 do arquivo mapeia o próprio MFT.

Os registros no MFT, apesar de possuírem o mesmo tamanho, apresentam formatos diferentes. Um registro tem um header, que identifica o registro, seguido por um ou mais atributos. Cada atributo é formado por um par de cabeçalho e valor, sendo que o cabeçalho identifica o que o valor representa. O NTFS define 13 tipos diferentes de atributos que podem aparecer em um registro, como nome do arquivo, descritor de segurança e dados do arquivo. No caso de arquivos pequenos, todos os seus atributos, inclusive os dados, podem estar presentes no próprio registro no MFT (Fig. 11). No entanto, para a maioria dos arquivos seus atributos podem ultrapassar facilmente o tamanho do registro, sendo colocados em clusters no disco (*nonresident attributes*).

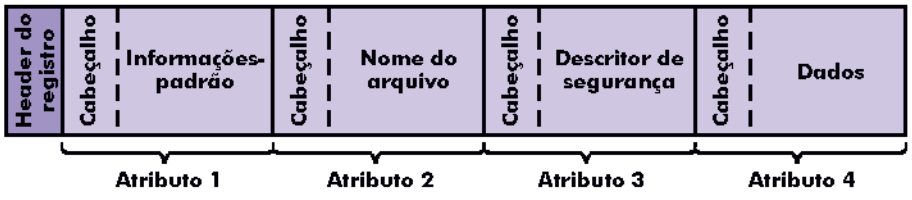


Fig. 11 Exemplo de registro para um pequeno arquivo.

Um arquivo em disco é formado por uma sequência de clusters não necessariamente contíguos no disco. Por questões de desempenho, sempre que possível o sistema tentará alocar todos os clusters que compõem o arquivo sequencialmente no disco. Quando não for possível, o arquivo será formado por vários conjuntos de clusters contíguos, sendo que cada conjunto é chamado de *extent*. O número de extents indica o grau de fragmentação do arquivo. Para mapear os extents

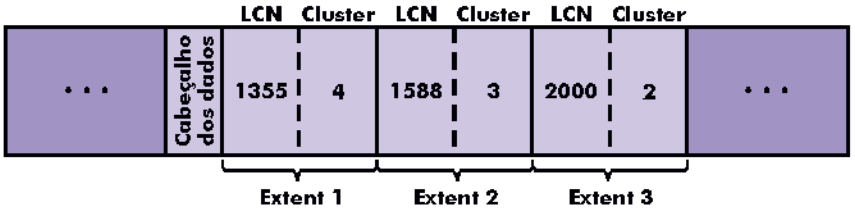


Fig. 12 Exemplo de registro de um arquivo.

de um arquivo, o NTFS registra no MFT a posição inicial do extent no disco, utilizando o LCN, e quantos clusters contíguos compõem o extent. A Fig. 12 ilustra o exemplo de um arquivo formado por três extents. Caso o número de extents do arquivo ultrapasse o tamanho do registro no MFT, um ou mais registros adicionais podem ser utilizados.

8. Gerência de Entrada/Saída

A gerência de entrada/saída é responsável por receber solicitações de E/S dos diversos threads e repassá-las aos diferentes tipos de dispositivos de E/S, como teclados, impressoras, monitores, discos, CD-ROMs, DVDs e mesmo a rede. O subsistema de E/S foi projetado de forma que novos dispositivos possam ser facilmente conectados ao sistema. Para isso, a gerência de E/S é estruturada em camadas e interage com outros subsistemas, estando intimamente ligada ao gerente de plug-and-play, de energia e de cache, além do sistema de arquivos (Fig. 13).

O MS Windows oferece diversas APIs relacionadas à gerência de E/S, sendo a maior parte ligada ao subsistema gráfico, chamado Graphics Device Interface (GDI). O GDI é um conjunto de rotinas que permite a uma aplicação manipular dispositivos gráficos, como monitores e impressoras, independentemente do tipo do dispositivo físico, funcionando como uma interface entre a aplicação e os drivers dos dispositivos. O GDI oferece funções de gerenciamento de janelas, menus, caixas de diálogo, cores, desenhos, textos, bitmaps, ícones e clipboard.

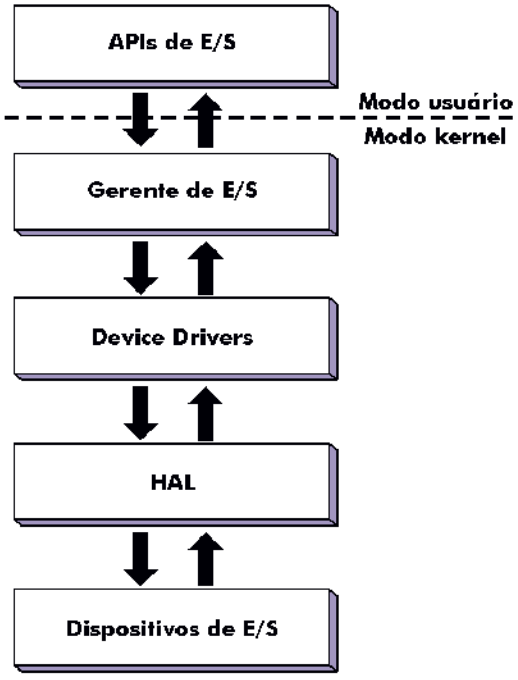


Fig. 13 Gerência de E/S.

O sistema oferece suporte a operações de E/S assíncronas, ou seja, um thread pode solicitar uma operação de E/S, continuar sua execução e depois tratar o término da solicitação. O término da operação pode ser sinalizada ao thread utilizando um dos diversos mecanismos de sincronização disponíveis no sistema.

O gerente de E/S (I/O Manager) é responsável por receber os pedidos de operações de E/S e repassá-los aos device drivers. Cada solicitação de E/S é representada por uma estrutura de dados chamada I/O Request Packet (IRP), que permite o controle de como a operação de E/S será processada. Quando uma operação de E/S é solicitada, o gerente de E/S cria o IRP e passa a estrutura para o device driver. Terminada a operação, o driver devolve o IRP para o gerente de E/S para completar a operação ou repassá-lo para um outro driver continuar o processamento.

Os device drivers no MS Windows são desenvolvidos a partir de um padrão chamado Windows Driver Model (WDM). O WDM define diversas características e funções que um driver deve oferecer para ser homologado pela Microsoft, como suporte a plug-and-play e a múltiplos processadores, gerência de energia e interface com os objetos do sistema operacional.

O MS Windows trabalha com diferentes tipos de drivers para implementar diversas funções, como a emulação de aplicações MS-DOS (virtual device drivers), interface com o subsistema gráfico GDI (display e printer drivers), implementação dos sistemas de arquivos (file system drivers), funções de filtragem (filter drivers) e controle de dispositivos de E/S (hardware device drivers).

Os device drivers podem ter acesso diretamente ao dispositivo de E/S, como em um driver de disco, ou podem trabalhar em camadas, separando funções ou implementando funções complementares. Por exemplo, as operações de compressão, criptografia e tolerância a falhas de discos são implementadas através de drivers específicos (filter driver) acima dos drivers ligados aos dispositivos de E/S. Os sistemas de arquivos NTFS e FAT também são implementados através de drivers especiais que recebem solicitações do gerente de E/S e as repassam para os drivers de disco correspondentes.

Unix

1. Histórico

Na década de 1960, inúmeros esforços foram direcionados para o desenvolvimento de um verdadeiro sistema operacional de tempo compartilhado que viesse a substituir os sistemas batch da época. Em 1965, o MIT (Massachusetts Institute of Technology), a Bell Labs e a General Electric se uniram para desenvolver o MULTICS (MULTiplexed Information and Computing Service). Em 1969, a Bell Labs retirou-se do projeto, porém um de seus pesquisadores nele envolvido, Ken Thompson, desenvolveu sua própria versão do sistema operacional, que veio a se chamar UNICS (UNiplexed Information and Computing Service) e, posteriormente, Unix.

O Unix foi inicialmente desenvolvido em Assembly para um minicomputador PDP-7 da Digital. Para torná-lo mais fácil de ser portado para outras plataformas, Thompson desenvolveu uma linguagem de alto nível chamada B e reescreveu o código do sistema nessa nova linguagem. Em função das limitações da linguagem B, Thompson e Dennis Ritchie, também da Bell Labs, desenvolveram a linguagem C, na qual o Unix seria reescrito e, posteriormente, portado para um minicomputador PDP-11 em 1973.

No ano seguinte, Ritchie e Thompson publicaram o artigo *The Unix Timesharing System* (Ritchie e Thompson, 1974), que motivou a comunidade acadêmica a solicitar uma cópia do sistema. Na época, a Bell Labs era uma subsidiária da AT&T e, mesmo tendo criado e desenvolvido o Unix, não podia comercializá-lo devido às leis americanas antimonopólio, que impediam seu envolvimento no mercado de computadores. Apesar dessa limitação, as universidades poderiam licenciar o Unix, recebendo inclusive o código-fonte do sistema. Como a grande maioria das universidades utilizava computadores da linha PDP-11, não existiam dificuldades para se adotar o sistema como plataforma-padrão no meio acadêmico.

Uma das primeiras instituições de ensino a licenciar o Unix foi a universidade de Berkeley, na Califórnia. A universidade desenvolveu sua própria versão do sistema, batizada de 1BSD (First Berkeley Software Distribution), seguida por outras versões, chegando até a 4.4BSD, quando o projeto acadêmico foi encerrado. O Unix de Berkeley introduziu inúmeros melhoramentos no sistema, merecendo destaque o mecanismo de memória virtual, C shell, Fast File System, sockets e o protocolo TCP/IP. Em função dessas facilidades, vários fabricantes passaram a utilizar

o BSD como base para seus próprios sistemas, como a Sun Microsystems e a Digital. Para dar continuidade ao desenvolvimento do Unix de Berkeley foi criada a Berkeley Software Design que, posteriormente, lançaria o sistema FreeBSD.

Em 1982, a AT&T foi autorizada a comercializar o sistema que tinha desenvolvido. A primeira versão lançada foi a System III, logo seguida pela System V. Diversas versões foram posteriormente desenvolvidas, sendo a versão System V Release 4 (SVR4) a que teve a maior importância. Vários fabricantes basearam seus sistemas no Unix da AT&T, como a IBM, a HP e a SCO. Em 1993, o Unix da AT&T é comercializado para a Novell, que, posteriormente, lança o UnixWare, com base nesse sistema. No mesmo ano, a Novell transfere os direitos sobre a marca Unix para o consórcio X/Open e, posteriormente, vende o UnixWare para a SCO.

Em 1991, o finlandês Linus Torvalds começou o desenvolvimento do Linux, com base em suas experiências com o sistema Minix. O Minix foi desenvolvido pelo professor Andrew Tanenbaum, da Universidade Vrije, na Holanda, com fins apenas educacionais, e pode ser obtido livremente, incluindo o código-fonte (Minix, 2002). O Linux evoluiu a partir da colaboração de vários programadores que ajudaram no desenvolvimento do kernel, utilitários e vários aplicativos. Atualmente, o Linux é utilizado para fins tanto acadêmicos como comerciais e pode ser obtido sem custos, acompanhado do seu código-fonte (Linux, 2002). No Brasil existe o sistema Tropix, desenvolvido pelos pesquisadores Oswaldo Vernet e Pedro Salenbauch, do Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro. O código-fonte do Tropix está disponível na Internet gratuitamente (Tropix, 2002).

Várias tentativas foram feitas visando unificar as versões do Unix de Berkeley (BSD) e da AT&T (System V), além das inúmeras outras implementações oferecidas pelo mercado. A AT&T publicou um conjunto de especificações, conhecidas como System V Interface Definition. Fabricantes ligados à vertente do Unix da AT&T fundaram a Unix International, enquanto os ligados ao Unix de Berkeley criaram a Open Software Foundation (OSF). Nenhuma dessas iniciativas resultou na padronização de um único Unix.

A mais importante tentativa de unificação do Unix nesse sentido foi dada pelo IEEE (Institute of Electrical and Electronics Engineers) através do seu comitê POSIX (Portable Operating System Unix). Como resultado desse trabalho, surgiu o padrão IEEE 1003.1, publicado em 1990, estabelecendo uma biblioteca-padrão de chamadas e um conjunto de utilitários que todo sistema Unix deveria oferecer. Em 1995, o X/Open cria o UNIX95, um programa para garantir uma especificação única do Unix. Posteriormente, o consórcio passa a chamar-se The Open Group e lança a terceira versão de sua especificação, incluindo o padrão POSIX e representantes da indústria.

A Fig. 1 apresenta a evolução das duas principais vertentes do Unix e alguns dos sistemas operacionais derivados das versões da AT&T e de Berkeley (Lévénez, 2002). Atualmente, diversas versões do Unix são encontradas não só no meio acadêmico, mas também sendo comercializadas por inúmeros fabricantes, como a Sun Microsystems (SunOS e Solaris), HP (HP-UX), IBM (AIX) e Compaq (Compaq Unix).

As implementações do sistema Unix variam conforme suas versões, plataformas de hardware e fabricantes, principalmente se comparadas às versões originais de Berkeley e da AT&T. No decorrer deste capítulo será apresentada uma visão geral do sistema, sem entrar em detalhes das diferentes implementações.

2. Características

O sistema operacional Unix é um sistema multiprogramável, multiusuário, que suporta múltiplos processadores e implementa memória virtual. Entre as muitas razões para explicar o sucesso alcançado pelo Unix incluem-se as características a seguir:

- Escrito em uma linguagem de alto nível, o que torna fácil a compreensão e alteração do seu código e a portabilidade para outras plataformas de hardware.

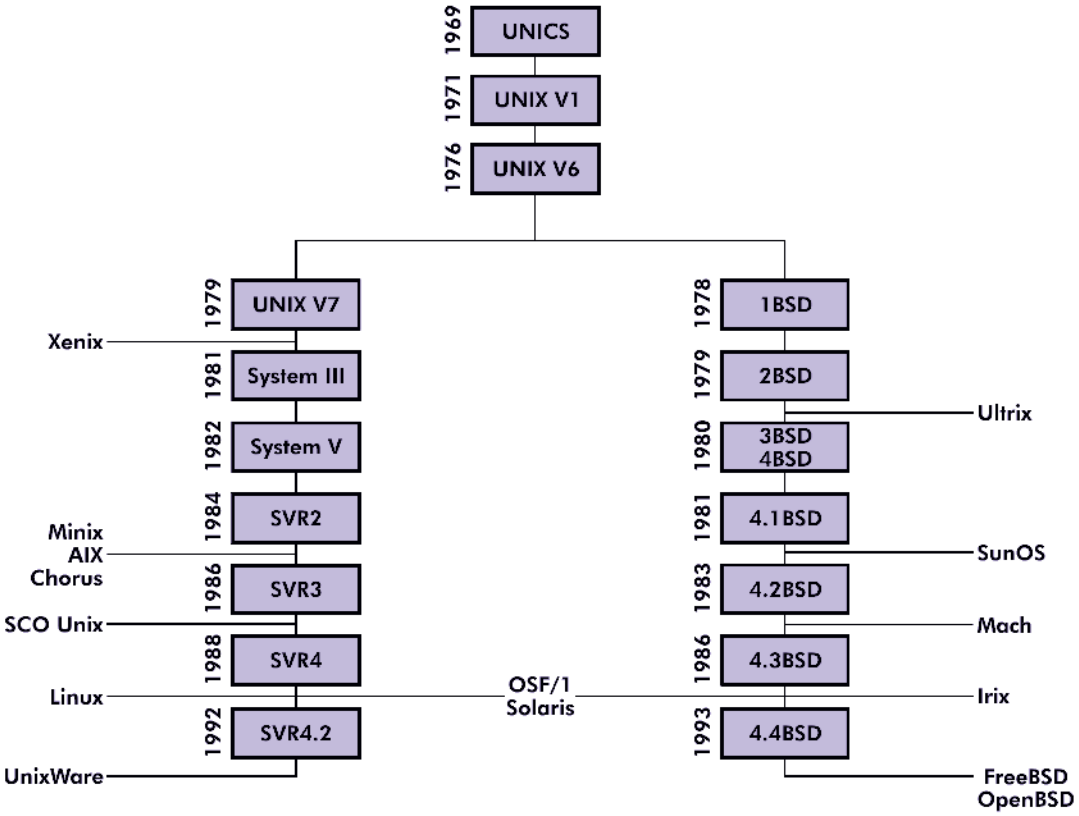


Fig. 1 Evolução do Unix.

- Oferece um conjunto de system calls que permite que programas complexos sejam desenvolvidos a partir de uma interface simples.
- Flexibilidade, podendo ser utilizado como sistema operacional de computadores pessoais, estações de trabalho e servidores de todos os portes voltados para banco de dados, Web, correio eletrônico e aplicação.
- Implementação de threads, em algumas versões, e diversos mecanismos de comunicação e sincronização, como memória compartilhada, pipes e semáforos.
- Suporte a um grande número de aplicativos disponíveis no mercado, sendo muitos gratuitos.
- Suporte a diversos protocolos de rede, como o TCP/IP, e interfaces de programação, como sockets, podendo ser utilizado como servidor de comunicação, roteador, firewall e proxy.
- Implementação de sistema de arquivos com uma estrutura bastante simples, em que os arquivos são representados apenas como uma sequência de bytes. Além disso, existem diversas opções para sistemas de arquivos distribuídos, como NFS (Network File System), AFS (Andrew File System) e DFS (Distributed File System).
- Oferece uma interface simples e uniforme com os dispositivos de E/S.

3. Estrutura do Sistema

A maior parte do código que compõe o núcleo do Unix é escrita em Linguagem C, e o restante, como os device drivers, em Assembly, o que confere ao sistema uma grande portabilidade para diferentes plataformas de hardware.

O Unix utiliza o modelo de camadas para a estruturação do sistema, implementando dois níveis de modo de acesso: usuário e kernel. A Fig. 2 apresenta as camadas do sistema de forma simplificada, sem a preocupação de representar a estrutura interna real e abranger a maioria das implementações.

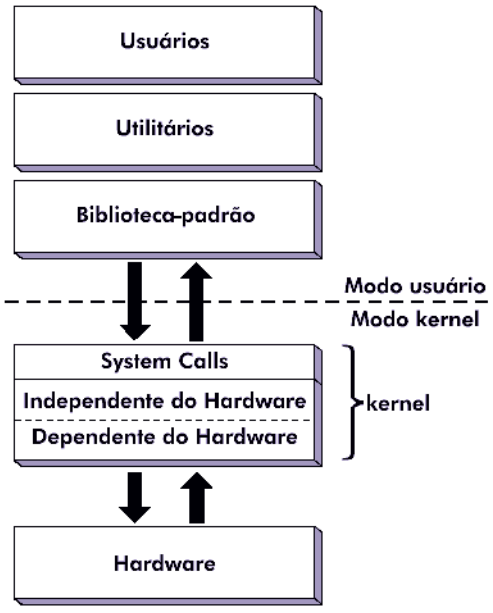


Fig. 2 Estrutura do Unix.

- **Kernel**

  O kernel é responsável por controlar o hardware e fornecer as system calls para que os programas tenham acesso às rotinas do sistema, como criação e gerência de processos, gerência de memória virtual, sistema de arquivos e gerência de E/S.

  O kernel pode ser dividido em duas partes: a parte dependente do hardware e a parte independente do hardware. A parte dependente do hardware consiste nas rotinas de tratamento de interrupções e exceções, device drivers, tratamento de sinais, ou seja, todo o código que deve ser reescrito quando se está portando um sistema Unix para uma nova plataforma. A parte independente do hardware não deve ter a princípio nenhum vínculo com a plataforma onde está sendo executada, e é responsável pelo tratamento das system calls, gerência de processos, gerência de memória, escalonamento, pipes, paginação, swapping e sistema de arquivos.

  O kernel do Unix, comparado com o de outros sistemas operacionais, oferece um conjunto relativamente pequeno de system calls, a partir das quais podem ser construídas rotinas de maior complexidade. A estratégia de criar um sistema modular e simples foi muito importante no desenvolvimento do Unix, pois novos utilitários podem ser facilmente integrados ao sistema sem que o kernel tenha que sofrer qualquer tipo de alteração.

- **Biblioteca-padrão**

  Para cada rotina do sistema existe um procedimento na biblioteca-padrão do Unix que permite esconder os detalhes da mudança de modo de acesso usuário-kernel-usuário. A biblioteca implementa uma interface entre os programas e o sistema operacional, fazendo com que as system calls sejam chamadas. O POSIX define a biblioteca-padrão e não as system calls, ou seja, quais procedimentos a biblioteca deve oferecer, as funções de cada procedimento e os parâmetros de entrada e saída que devem ser utilizados.

- **Utilitários**

  A camada mais externa do sistema é a interface com o usuário, formada por diversos programas utilitários, como editores de textos, compiladores e o shell. O shell é o interpretador de

comandos, responsável por ler os comandos do usuário, verificar se a sintaxe está correta e passar o controle para outros programas que realizam a tarefa solicitada. A Tabela 1 apresenta alguns exemplos de comandos e utilitários encontrados na maioria dos sistemas Unix.

Tabela 1 Exemplos de comandos e utilitários

Utilitário	Descrição
ls	Lista o conteúdo de diretórios.
cp	Copia arquivos.
mkdir	Cria um diretório.
pwd	Exibe o diretório corrente.
grep	Pesquisa por um determinado padrão dentro do arquivo.
sort	Ordena um arquivo.
cc	Executa o compilador C.
vi	Permite a criação e a edição de arquivos-textos, como programas e scripts.

Devido às várias implementações do Unix, três interpretadores de comandos se tornaram populares. O Bourne Shell (sh) foi o primeiro shell disponível, e pode ser encontrado em todas as versões do Unix. O C Shell (csh) é o padrão para o BSD, e o Korn Shell (ksh) é o padrão para o System V. Além das interfaces orientadas a caractere, existem também interfaces gráficas disponíveis, como o X Windows System.

4. Processos e Threads

O Unix, como um sistema multiprogramável, suporta inúmeros processos, que podem ser executados concorrentemente ou simultaneamente. O modelo de processo implementado pelo sistema é muito semelhante ao apresentado no Cap. 5 — Processo. As primeiras versões do Unix não implementavam o conceito de threads, porém as versões mais recentes já oferecem algum tipo de suporte a aplicações multithread.

Um processo é criado através da system call fork. O processo que executa o fork é chamado de processo-pai, enquanto o novo processo é chamado de processo-filho ou subprocesso. Cada processo-filho tem seu próprio espaço de endereçamento individual, independente do processo-pai. Apesar de o espaço de endereçamento dos subprocessos ser independente, todos os arquivos abertos pelo pai são compartilhados com seus filhos. Sempre que um processo é criado, o sistema associa identificadores, que fazem parte do contexto de software, permitindo implementar mecanismos de segurança (Tabela 2).

Tabela 2 Identificadores

Identificador	Descrição
PID	O Process Identification identifica unicamente um processo para o sistema.
PPID	O Parent Process Identification identifica o processo-pai.
UID	O User Identification identifica o usuário que criou o processo.
GID	O Group Identification identifica o grupo do usuário que criou o processo.

A system call fork, além de criar o subprocesso, copia o espaço de endereçamento do processo-pai para o filho, incluindo o código executável e suas variáveis. Por ser apenas uma cópia, uma alteração no espaço de endereçamento do processo-filho não implica modificação das posições de memória do processo-pai. As versões mais recentes do Unix utilizam a técnica conhecida como copy-on-write para evitar a duplicação de todo o espaço de endereçamento do processo-pai e o tempo gasto na tarefa. Nesse esquema, o espaço de endereçamento do processo-pai é comparti-

lhado com o filho, sendo copiadas apenas as páginas que foram alteradas pelo subprocesso. Uma outra solução, implementada no Unix BSD, é a utilização da system call vfork, que não copia o espaço de endereçamento do processo-pai.

Quando o sistema é ativado o processo 0 é criado, o qual, por sua vez, cria o processo 1. Esse processo, conhecido como init, é o pai de todos os outros processos que venham a ser criados no sistema. Sempre que um usuário inicia uma sessão, o processo init cria um novo processo para a execução do shell. Quando o usuário executa um comando, dois eventos podem ocorrer: o shell pode criar um subprocesso para a execução do comando ou o próprio shell pode executar o comando no seu próprio contexto.

No Unix é possível criar processos foreground e background. No primeiro caso, existe uma comunicação direta do usuário com o processo durante a sua execução. Processos background não podem ter interação com o usuário e são criados com o uso do símbolo &. O comando a seguir mostra a criação de um processo background para a execução de prog.

```
# pgm &
```

Processos do sistema operacional no Unix são chamados de daemons. Os daemons são responsáveis por tarefas administrativas no sistema, como, por exemplo, escalonamento de tarefas (cron), gerência de filas de impressão, suporte a serviços de rede, suporte à gerência de memória (swapper) e gerência de logs. Os daemons são criados automaticamente durante a inicialização do sistema.

Processos no Unix podem se comunicar através de um mecanismo de troca de mensagens, conhecido como pipe. O comando a seguir mostra o mecanismo de pipe entre dois processos. O primeiro processo é criado a partir da execução do comando ls, que lista os arquivos do diretório corrente. A saída desse processo é redirecionada para a entrada do segundo processo, criado para a execução do comando grep. O comando grep seleciona dentre a lista de arquivos as linhas que possuem o string “pgm”.

```
# ls | grep pgm
```

Outro mecanismo de comunicação entre processos muito importante no Unix é conhecido como sinal. Um sinal permite que um processo seja avisado da ocorrência de eventos síncronos ou assíncronos. Por exemplo, quando um programa executa uma divisão por zero, o sistema avisa ao processo sobre o problema através de um sinal. O processo, por sua vez, pode aceitar o sinal ou simplesmente ignorá-lo. Caso o processo aceite o sinal, é possível especificar uma rotina de tratamento.

Sinais são definidos de diferentes maneiras em cada versão do Unix. O POSIX define um conjunto de sinais padrões que devem ser suportados pelo Unix, a fim de compatibilizar a utilização de sinais. A Tabela 3 apresenta alguns dos sinais definidos pelo POSIX.

Tabela 3 Sinais POSIX

Sinal	Descrição
SIGALRM	Sinaliza o término de um temporizador.
SIGFPE	Sinaliza um erro em uma operação de ponto flutuante.
SIGILL	Sinaliza que a tecla DEL ou CTRL-C foi digitada para interromper o processo em execução.
SIGKILL	Sinaliza que o processo deve ser eliminado.
SIGTERM	Sinaliza que o processo deve terminar.
SIGSEGV	Sinaliza que o processo fez acesso a um endereço inválido de memória.
SIGUSR1	Pode ser definido pela própria aplicação.
SIGUSR2	Pode ser definido pela própria aplicação.

Um processo no Unix é formado por duas estruturas de dados: a estrutura do processo (proc structure) e a área do usuário (user area ou u area). A estrutura do processo, que contém o seu contexto de software, deve ficar sempre residente na memória principal, enquanto a área do usuário pode ser retirada da memória, sendo necessária apenas quando o processo é executado. A Tabela 4 apresenta um resumo das informações contidas nessas duas estruturas.

Tabela 4 Processo no Unix

Estrutura do processo	Área do usuário
Identificação: PID, PPID, GID e UID	Registradores
Estado do processo e prioridade	Tabela de descritores de arquivos
Endereço da área do usuário	Contabilidade de recursos do sistema, como UCP, memória e disco
Máscara de sinais	Informações sobre a system call corrente
Ponteiros para as tabelas de páginas	Tratadores de sinais

Os processos existentes no sistema são organizados em um vetor, chamado tabela de processos, onde cada elemento representa uma estrutura do processo. O tamanho desse vetor é predefinido e limita o número máximo de processos no sistema. A estrutura do processo, por sua vez, possui um ponteiro para a área do usuário. Quando um processo executa um fork, o sistema procura por um elemento livre na tabela de processos, onde é criada a estrutura do processo-filho, a partir das informações copiadas da estrutura do processo-pai.

A Tabela 5 apresenta algumas system calls voltadas para a gerência de processos disponíveis na maioria dos sistemas Unix.

Tabela 5 Gerência de processos

System call	Descrição
fork	Cria um processo-filho idêntico ao processo-pai.
execve execv execl execle	Permite substituir o código executável do processo-filho depois da sua criação.
waitpid	Aguarda até o término do processo-filho.
exit	Termina o processo corrente.
kill	Envia um sinal para um processo.
sigaction	Define qual ação deve ser tomada ao receber um determinado sinal.
alarm	Permite definir um temporizador.

Em função do overhead gerado no mecanismo de criação e eliminação de processos, vários sistemas Unix implementaram o conceito de threads, porém sem qualquer preocupação com compatibilidade (Tabela 6). Em 1995, o padrão POSIX P1003.1c, também conhecido como Pthreads, foi aprovado, permitindo que aplicações multithread pudessem ser desenvolvidas de forma padronizada.

Tabela 6 Arquitetura de threads

Ambiente	Arquitetura
Compaq Unix 5	Modo híbrido
IBM-AIX 4.2	Modo kernel
HP-UX 10.1	Modo usuário
Linux 2	Modo kernel
Sun Solaris 8	Modo híbrido
SunOS 4	Modo usuário

O POSIX não define como os threads devem ser implementados no sistema, ou seja, o padrão pode ser implementado utilizando pacotes apenas em modo usuário, modo kernel ou uma combinação de ambos (modo híbrido). As vantagens e desvantagens de cada tipo de implementação podem

ser consultadas no Cap. 6 — Thread. O padrão POSIX também define mecanismos de sincronização entre threads, como semáforos, mutexes e variáveis condicionais. A Tabela 7 apresenta as principais system calls definidas pelos Pthreads.

Tabela 7 Gerência de POSIX threads

System call	Descrição
pthread_create	Cria um novo thread.
pthread_exit	Finaliza o thread corrente.
pthread_join	Aguarda pelo término de um thread.
pthread_mutex_init	Cria um mutex.
pthread_mutex_lock	Verifica o estado do mutex.
pthread_mutex_unlock	Libera o mutex.
pthread_mutex_destroy	Elimina o mutex.
pthread_cond_init	Cria uma variável condicional.
pthread_cond_wait	Aguarda por uma variável condicional.
pthread_cond_signal	Libera um thread.
pthread_cond_destroy	Elimina uma variável condicional.

5. Gerência do Processador

A gerência do processador no Unix utiliza dois tipos de política de escalonamento: escalonamento circular com prioridades e escalonamento por prioridades. A política de escalonamento tem o objetivo de permitir o compatilhamento da UCP por vários processos interativos e batch, além de oferecer baixos tempos de respostas para os usuários interativos.

Os processos no Unix podem ter prioridades entre 0 e 127, e quanto menor o valor, maior a prioridade. Processos executados no modo usuário têm valor de prioridade entre 50 e 127 (menor prioridade), enquanto processos no modo kernel têm valores de prioridade entre 0 e 49 (maior prioridade). Os processos no estado de pronto ficam aguardando para serem escalonados em diversas filas, cada fila associada a uma prioridade. O algoritmo de escalonamento seleciona para execução o processo de maior prioridade, ou seja, o primeiro processo da fila de menor valor (Fig. 3). O processo, depois de escalonado, poderá permanecer no processador no máximo uma fatia de tempo, que varia entre 10 e 100 milissegundos. Depois de terminado seu quantum, o processo retorna para o final da fila associada à sua prioridade.

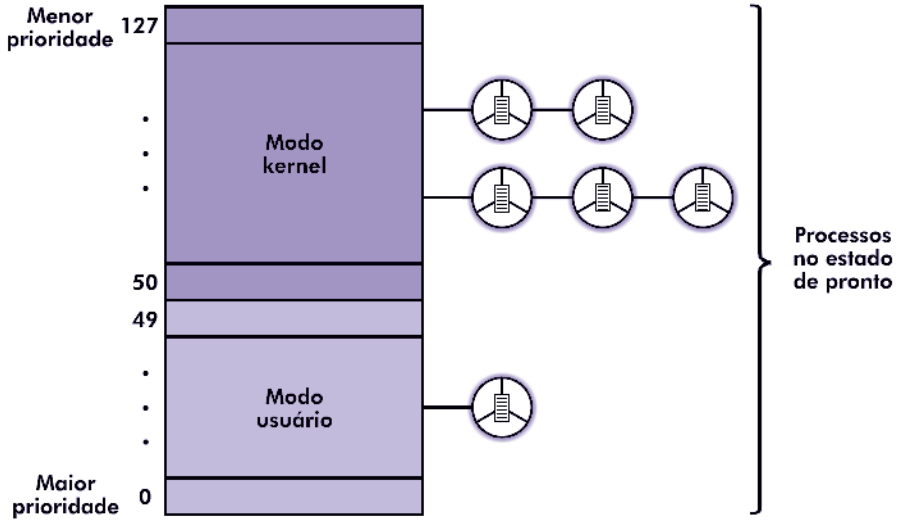


Fig. 3 Níveis de prioridade.

O escalonador recalcula a prioridade de todos os processos no estado de pronto periodicamente. Para realizar o cálculo da nova prioridade é utilizada a fórmula a seguir, com base em três variáveis: p_cpu, p_nice e base.

$$\text{Prioridade} = \text{p\_cpu} + \text{p\_nice} + \text{base}$$

A variável p_cpu pertence ao contexto de software do processo e permite contabilizar o tempo durante o qual o processo utilizou o processador. Quando o processo é criado, a variável é inicializada com zero. Sempre que o processo é executado, o valor de p_cpu é incrementado de uma unidade a cada tick do clock, até o valor máximo de 127. Quanto maior o valor da variável, menor será sua prioridade. A variável p_cpu permite penalizar os processos CPU-bound e, dessa maneira, distribuir de forma mais igualitária o processador.

O escalonador, além de recalcular a prioridade dos processos, também reduz o valor de p_cpu periodicamente, com base no valor de decay. O cálculo de decay varia conforme a versão do Unix e tem a função de evitar o problema de starvation de processos de baixa prioridade. Quanto menos um processo utilizar o processador, menor será o valor de p_cpu e, logo, maior sua prioridade. Esse esquema também privilegia processos I/O-bound, pois mantém suas prioridades elevadas, permitindo que tenham maiores chances de serem executados quando saírem do estado de espera.

No BSD, a variável p_nice pode assumir valores entre 0 e 39, sendo o default 20. Quanto maior o valor atribuído à variável, menor a prioridade do processo. A variável permite alterar a prioridade de um processo de diferentes maneiras. A própria aplicação pode reduzir voluntariamente a sua prioridade a fim de não prejudicar os demais processos utilizando a system call nice. O administrador do sistema pode reduzir o valor da variável utilizando o comando nice, aumentando assim a prioridade de um processo. Processos em background recebem automaticamente um valor maior para p_nice, a fim de não prejudicar os processos interativos.

A variável-base geralmente está associada ao tipo de evento que colocou o processo no estado de espera. Quando a espera termina, é atribuído um baixo valor à variável, fazendo com que o processo receba um aumento de prioridade. Com isso, processos I/O-bound têm suas prioridades aumentadas dependendo do tipo de operação realizada, fazendo com que processos interativos não sejam prejudicados. Por outro lado, processos CPU-bound podem ser executados enquanto os processos I/O-bound estão no estado de espera.

6. Gerência de Memória

As primeiras versões do Unix utilizavam basicamente a técnica de swapping para a gerência de memória. Apenas a partir da versão 3BSD o Unix passou a utilizar paginação por demanda. Atualmente, a grande maioria das versões do Unix, tanto BSD como System V, implementa gerência de memória virtual por paginação com swapping. Neste item será apresentada uma abordagem com base na versão 4BSD, mas grande parte dos conceitos e mecanismos apresentados também pode ser encontrada no System V.

No Unix, os conceitos de espaço de endereçamento virtual e mapeamento seguem as mesmas definições apresentadas no Cap. 10 — Gerência de Memória Virtual. Muitos detalhes de implementação, como tamanho de página, níveis de tabelas de páginas e TLBs, são dependentes da arquitetura de hardware, podendo variar conforme a versão de cada sistema.

O espaço de endereçamento dos processos no Unix é dividido em três segmentos: texto, dados e pilha (Fig. 4). O segmento de texto corresponde à área onde está o código executável dos programas, sendo uma área protegida contra gravação. O segmento de texto é estático e pode ser compartilhado por vários processos, utilizando o esquema de memória compartilhada. O segmento

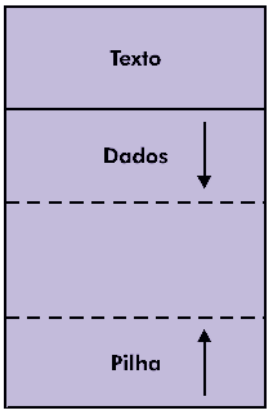


Fig. 4 Espaço de endereçamento.

de dados corresponde às variáveis do programa, como tipos numéricos, vetores e strings. A área de dados é dinâmica, podendo aumentar ou diminuir durante a execução do programa. A pilha armazena informações de controle do ambiente do processo, como parâmetros passados a um procedimento ou system call. A área de pilha cresce dinamicamente do endereço virtual mais alto para o mais baixo.

O Unix implementa o esquema de paginação por demanda como política de busca de páginas. Nesse esquema, páginas do processo são trazidas do disco para a memória principal apenas quando são referenciadas. O sistema mantém uma lista de páginas livres com todos os frames disponíveis na memória, e gerencia os frames de todos os processos em uma lista de páginas em uso. Quando um processo faz referência a uma página que não se encontra na lista de páginas em uso, ocorre um page fault. O sistema identifica se a página está na memória através do bit de validade. Nesse caso, a gerência de memória retira uma página da lista de páginas livres e transfere para a lista de páginas em uso. Como pode ser observado, o Unix utiliza uma política global de substituição de páginas.

O mecanismo de paginação é implementado parte pelo kernel e parte pelo daemon page (processo 2). Periodicamente, o daemon é ativado para verificar o número de páginas livres na memória. Se o número de páginas for insuficiente, o daemon page inicia o trabalho de liberação de páginas dos processos para recompor a lista de páginas livres. As páginas de texto podem ser transferidas sem problemas para a lista de páginas livres, pois podem ser recuperadas no arquivo executável. Por outro lado, para as páginas de dados o sistema utiliza o bit de modificação para verificar se a página foi modificada. Nesse caso, antes de ser liberada para a lista de páginas livres a página é gravada em disco.

O sistema implementa uma variação do algoritmo FIFO circular como política de substituição de páginas. Esse algoritmo, conhecido como two-handed clock, utiliza dois ponteiros, ao contrário do FIFO circular, que implementa apenas um. O primeiro ponteiro fica à frente do segundo um certo número de frames na lista de páginas em uso. Enquanto o primeiro ponteiro desliga o bit de referência das páginas, o segundo verifica o seu estado. Se o bit de referência continuar desligado, significa que a página não foi referenciada desde o momento em que o primeiro ponteiro desligou o bit, fazendo com que essa página seja selecionada. Apesar de estarem liberadas para outros processos, as páginas selecionadas permanecem um certo tempo intactas na lista de páginas livres. Dessa forma, é possível que as páginas retornem aos processos de onde foram retiradas, eliminando-se a necessidade de acesso a disco. O daemon page também é responsável por implementar a política de substituição de páginas.

Em casos em que o sistema não consegue manter um número suficiente de páginas livres, o mecanismo de swapping é ativado. Nesse caso, o daemon swapper seleciona, inicialmente, os

processos que estão há mais tempo inativos. Em seguida, o swapper seleciona dentre os quatro processos que mais consomem memória principal aquele que estiver mais tempo inativo. Esse mecanismo é repetido até que a lista de páginas livres retorne ao seu tamanho normal. Para cada processo transferido para disco é atribuída uma prioridade, calculada em função de vários parâmetros. Em geral, o processo que está mais tempo em disco é selecionado para retornar à memória principal.

Para controlar todas as páginas e listas na memória principal, a gerência de memória mantém uma estrutura de mapeamento dos frames na memória (core map), com informações sobre todas as páginas livres e em uso. O core map fica residente na parte não paginável da memória principal, juntamente com o kernel do sistema.

7. Sistema de Arquivos

O sistema de arquivos foi o primeiro componente a ser desenvolvido no Unix, e as primeiras versões comerciais utilizavam o System V File System (S5FS). Devido às suas limitações, Berkeley desenvolveu o Fast File System (FFS) introduzido no 4.2BSD. Posteriormente, o SVR4 passou também a suportar o sistema de arquivos de Berkeley.

Um arquivo no Unix é simplesmente uma sequência de bytes sem significado para o sistema operacional, que desconhece se o conteúdo do arquivo representa um texto ou um programa executável. O sistema tem apenas a função de prover o acesso sequencial ou aleatório ao arquivo, ficando a cargo da aplicação a organização e outros métodos de acesso. O tamanho do nome de arquivos era, inicialmente, limitado a 14 caracteres, mas as versões mais recentes ampliam esse nome para 255 caracteres.

O sistema de arquivos do Unix tem como base uma estrutura de diretórios hierárquica, sendo o diretório raiz (root) representado pela barra (/). Os diretórios são implementados através de arquivos comuns, responsáveis pela manutenção da estrutura hierárquica do sistema de arquivos. Todo diretório contém os nomes de arquivos ponto (.) e dois pontos (..), que correspondem, respectivamente, ao próprio diretório e ao seu diretório-pai (Fig. 5).

Alguns nomes de diretório do sistema de arquivos são padronizados, como o diretório de programas executáveis do sistema (/bin), o diretório de arquivos especiais ligados aos dispositivos de E/S (/dev), o diretório de bibliotecas (/lib) e o diretório que agrupa os subdiretórios dos usuários (/usr). Geralmente, cada usuário possui seu diretório default de login, denominado diretório home ou de trabalho (/maia e /machado).

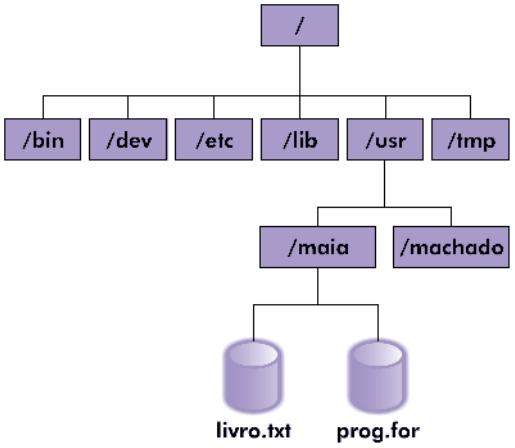


Fig. 5 Estrutura de diretórios.

A localização de um arquivo dentro da estrutura de diretórios é indicada utilizando-se um pathname, que representa uma sequência de diretórios separados por barra. Existem dois tipos de pathname: absoluto e relativo. Um pathname absoluto indica a localização de um arquivo através de uma sequência completa de diretórios e subdiretórios, a partir da raiz do sistema de arquivos. Um pathname relativo indica a localização de um arquivo a partir do diretório corrente. Por exemplo, na Fig. 5 o pathname /usr/maia/livro.txt representa o caminho absoluto para o arquivo livro.txt. Caso um usuário esteja posicionado no diretório /usr/machado, basta especificar o caminho relativo ../maia/livro.txt para ter acesso ao mesmo arquivo.

O sistema de arquivos proporciona um mecanismo para compartilhamento de arquivos conhecido como link simbólico. Um link é uma entrada em um diretório que faz referência a um arquivo em um outro diretório. Um arquivo pode possuir múltiplos links, de forma que diferentes nomes de arquivos podem ser utilizados por diversos usuários no acesso às informações de um único arquivo. O número de links de um arquivo é o número de diferentes nomes que ele possui. Existem diversas vantagens na utilização de links, como redução de utilização de espaço em disco, uma vez que existe uma única cópia dos dados, compartilhamento entre diversos usuários da última versão do arquivo e facilidade de acessso a arquivos em outros diretórios.

Cada arquivo no Unix pertence a uma ou mais dentre três categorias de usuários. Todo arquivo ou diretório tem um dono (user) e pertence a um grupo (group). Qualquer usuário que não seja o dono do arquivo e não pertença ao grupo enquadra-se na categoria outros (others). O administrador do sistema, utilizando a conta root, não pertence a nenhuma dessas categorias, tendo acesso irrestrito a todos os arquivos. Para cada categoria de usuário podem ser concedidos três tipos de acesso: leitura (r), gravação (w) e execução (x). A Tabela 8 apresenta as permissões que podem ser aplicadas a arquivos e diretórios.

Tabela 8 Permissões para arquivos e diretórios

Arquivo	Descrição
r	Permissão para ler e copiar o arquivo.
w	Permissão para alterar e eliminar o arquivo.
x	Permissão para executar o arquivo.
Diretório	**Descrição**
r	Permissão para listar o conteúdo do diretório.
w	Permissão para criar, eliminar e renomear arquivos no diretório.
x	Permissão para que o usuário possa se posicionar no diretório e acessar os arquivos abaixo desse diretório.

A Tabela 9 apresenta algumas system calls relacionadas à gerência do sistema de arquivos, envolvendo operações com arquivos e diretórios.

Tabela 9 System calls do sistema de arquivos

System call	Descrição
creat	Cria um arquivo.
open	Abre um arquivo.
read	Lê um dado do arquivo para o buffer.
write	Grava um dado do buffer no arquivo.
position	Posiciona o ponteiro do arquivo.
close	Fecha um arquivo.
mkdir	Cria um diretório.
chdir	Altera o diretório default.
rmdir	Elimina um diretório.
link	Cria um link simbólico para um arquivo.
unlink	Elimina um link simbólico para um arquivo.

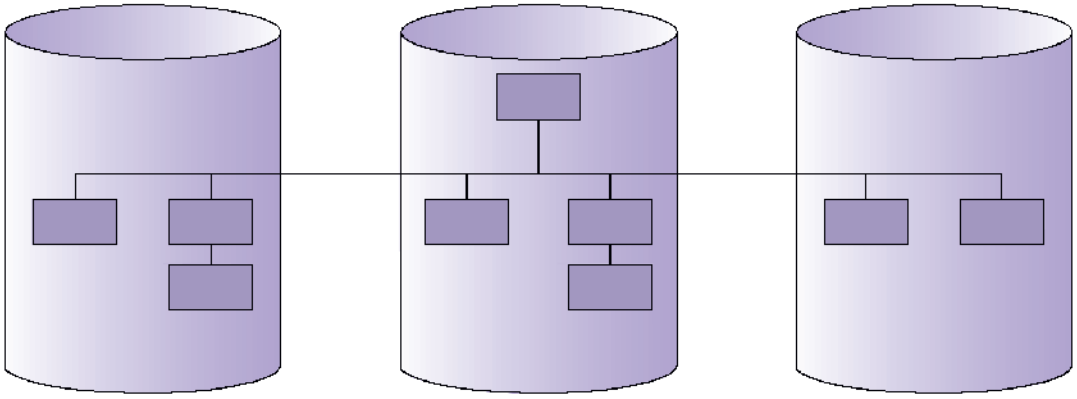


Fig. 6 Sistema de arquivos.

No Unix não existe uma dependência entre a estrutura lógica do sistema de arquivos e o local onde os arquivos estão fisicamente armazenados (Fig. 6). Dessa forma, é possível criar um sistema de arquivos onde os diretórios e arquivos estão fisicamente distribuídos em vários discos, porém para o usuário é como se existisse uma única estrutura lógica de diretórios. Esse modelo permite adicionar novos discos ao sistema de arquivos sempre que necessário, sem alterar sua estrutura lógica. Além disso, os diversos discos podem estar residentes em estações remotas. Nesse caso, existem padrões para a implementação de sistemas de arquivos remotos, como Network File System (NFS), Remote File System (RFS) e Andrew File System (AFS).

A estrutura do sistema de arquivos do Unix varia conforme a implementação. Em geral, qualquer disco deve ter a estrutura semelhante à descrita na Fig. 7. O boot block, quando utilizado, serve para realizar a carga do sistema. O super block possui informações sobre a estrutura do sistema de arquivos, incluindo o número de i-nodes, o número de blocos do disco e o início da lista de blocos livres. Qualquer problema com o super block tornará inacessível o sistema de arquivos.

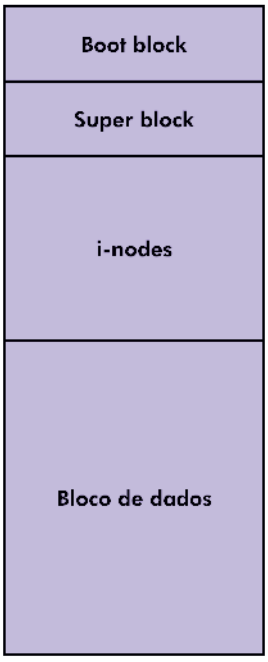


Fig. 7 Estrutura do sistema de arquivos.

Os i-nodes (index-nodes) permitem identificar e mapear os arquivos no disco. Cada i-node possui 64 bytes e descreve um único arquivo contendo seus atributos, como tamanho, datas de criação e modificação, seu dono, grupo, proteção, permissões de acesso, tipo do arquivo, além da localização dos blocos de dados no disco. Os blocos de dados contêm os dados propriamente ditos, ou seja, arquivos e diretórios. Um arquivo pode ser formado por um ou mais blocos, contíguos ou não no disco.

No caso de arquivos pequenos, todos os blocos que compõem o arquivo podem ser mapeados diretamente pelo i-node. No caso de arquivos grandes, o número de entradas para endereçamento no i-node não é suficiente para mapear todos os blocos do arquivo. Nesse caso, utilizam-se os redirecionamentos único, duplo e triplo. No redirecionamento único uma das entradas no i-node aponta para uma outra estrutura, que por sua vez endereça os blocos do arquivo no disco. Os redirecionamentos duplo e triplo são apenas uma extensão do conceito apresentado (Fig. 8).

8. Gerência de Entrada/Saída

A gerência de entrada/saída no Unix foi desenvolvida de forma integrada ao sistema de arquivos. O acesso aos dispositivos de E/S, como terminais, discos, impressoras e à própria rede, é feito através de arquivos especiais. Cada dispositivo está associado a um ou mais arquivos especiais, localizados no diretório /dev. Por exemplo, uma impressora pode ser o arquivo /dev/1p; um terminal, /dev/tty1; e uma interface de rede, /dev/net.

Os arquivos especiais podem ser acessados da mesma forma que qualquer outro arquivo, utilizando simplesmente as system calls de leitura e gravação. No Unix, todas as operações de E/S são realizadas como uma sequência de bytes, não existindo o conceito de registro ou método de acesso. Isso permite enviar o mesmo dado para diferentes dispositivos de saída, como um arquivo

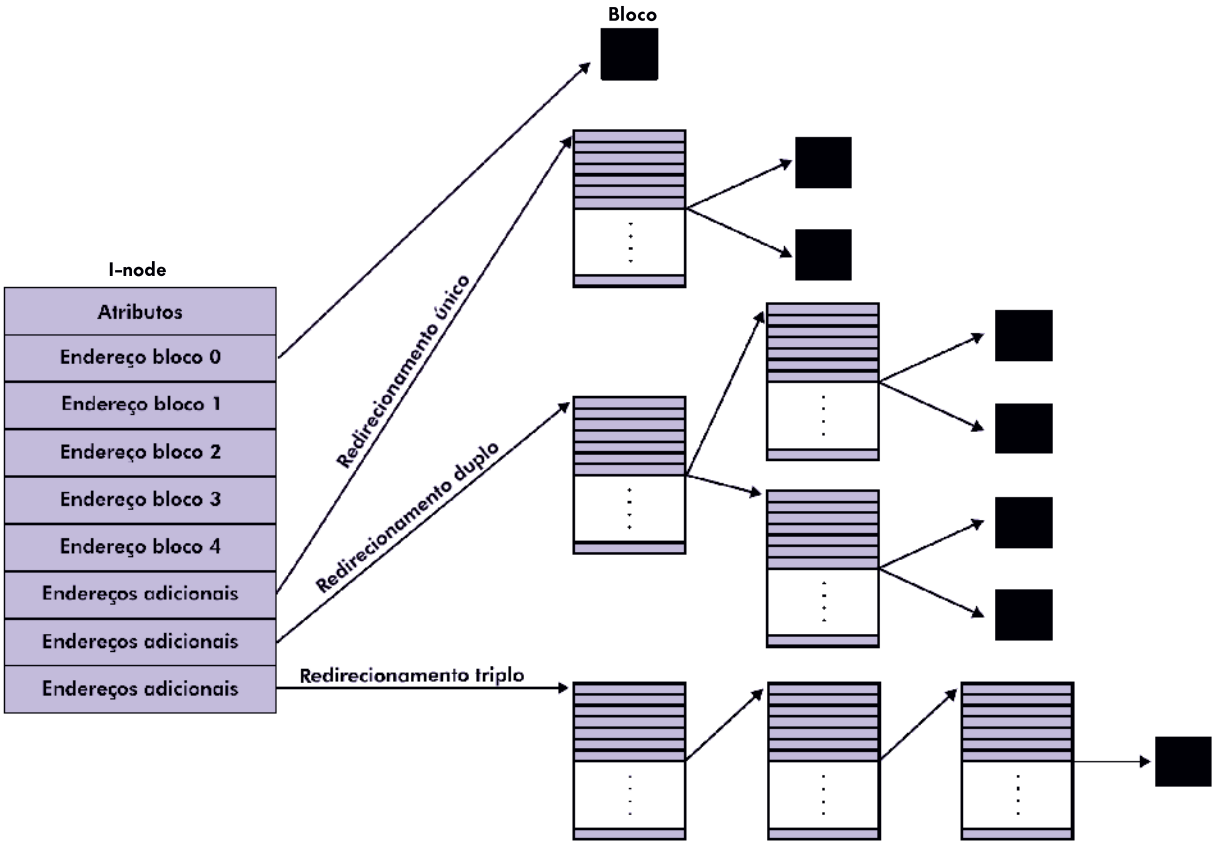


Fig. 8 Estrutura do i-node.

em disco, terminal, impressora ou linha de comunicação. Dessa forma, as system calls de E/S podem manipular qualquer tipo de dispositivo de maneira uniforme.

A Fig. 9 apresenta as camadas que compõem a gerência de E/S no Unix. Os processos se comunicam com o subsistema de E/S através das system calls de E/S. O subsistema de E/S é a parte do kernel responsável por lidar com as funções entrada e saída independentes do dispositivo, como buffering e controle de acesso. Para permitir a comunicação entre o subsistema de E/S e os diferentes drivers de maneira uniforme, o sistema implementa uma interface com os device drivers, padronizada com base nas especificações Device Driver Interface (DDI) e Driver Kernel Interface (DKI).

Os device drivers têm a função de isolar os dispositivos de E/S do restante do kernel, tornando-o independente da arquitetura de hardware, e para cada dispositivo existe um device driver associado. Os device drivers são acoplados ao sistema operacional quando o kernel é gerado, e sempre que um novo dispositivo é acrescentado ao sistema o driver correspondente deve ser acoplado ao núcleo. A tarefa de geração do kernel não é simples, e exige que o sistema seja reinicializado. As versões mais recentes do Unix, como o Linux, permitem que os device drivers possam ser acoplados ao núcleo com o sistema em funcionamento, sem a necessidade de uma nova geração do kernel e reinicialização do sistema.

Os device drivers podem ser divididos em dois tipos: orientados a bloco e orientados a caractere. Os device drivers orientados a bloco estão ligados a dispositivos como discos e CD-ROMs, que permitem a transferência de blocos de informações do mesmo tamanho. Os

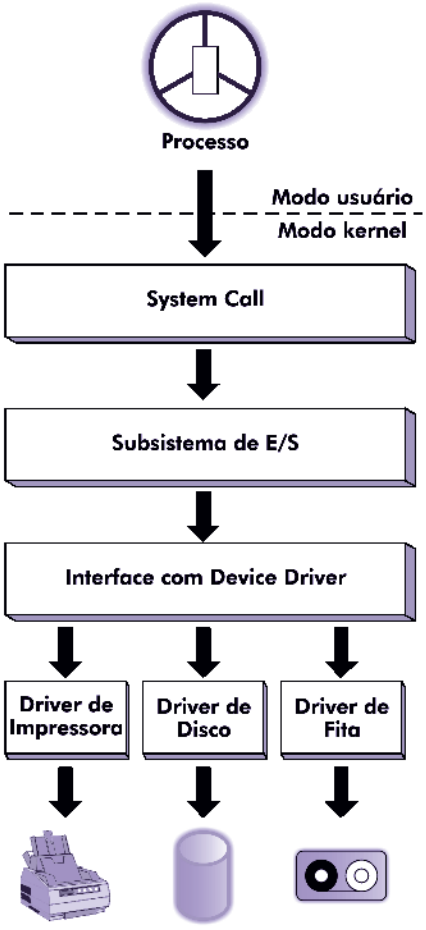


Fig. 9 Gerência de E/S.

drivers orientados a caractere são voltados para atender dispositivos como terminais e impressoras que transferem informação de tamanho variável, geralmente caractere a caractere ou uma sequência de caracteres.

No caso das operações orientadas a bloco, deve existir a preocupação em minimizar o número de transferências entre o dispositivo e a memória, utilizando o buffer cache (Fig. 10). O buffer cache é uma área na memória principal onde ficam armazenados temporariamente os blocos recentemente referenciados. Por exemplo, quando uma operação de leitura a disco é realizada o subsistema de E/S verifica se o bloco está no buffer cache. Se o bloco se encontra no cache, é possível passá-lo diretamente para o sistema de arquivos, sem acesso ao disco, melhorando assim o desempenho do sistema.

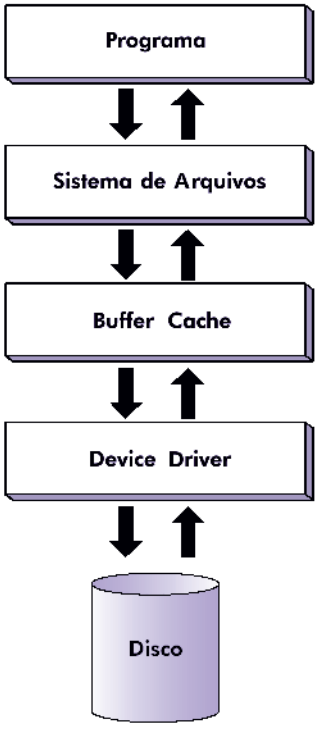


Fig. 10 Operação orientada a bloco.

OpenVMS

1. Histórico

No final da década de 1960, a Digital Equipment Corporation (DEC) anunciou o lançamento de sua linha de computadores com base no processador PDP-11. A linha PDP alcançou grande sucesso comercial na década de 1970, permitindo que a Digital se tornasse uma das líderes de mercado ao introduzir o conceito de minicomputadores. A relação preço/desempenho dessas máquinas era bastante superior à dos mainframes que predominavam no cenário da época.

Em 1973, a Digital designou o engenheiro David Cutler para projetar um sistema operacional de tempo real, denominado RSX-11M, para a plataforma PDP-11. Este sistema já possuía conceitos avançados para a época, como sistema de arquivos hierárquicos, utilização da técnica de swapping e um conjunto de ferramentas de apoio a desenvolvimento de sistemas.

Apesar do sucesso, o PDP-11 possuía uma séria limitação na sua capacidade de endereçamento, restrita a 16 bits. Com base neste fato, a Digital investiu na arquitetura VAX (Virtual Address eXtension) de 32 bits, visando oferecer um novo processador com capacidade de endereçamento que satisfizesse seus clientes por um longo período de tempo. Assim, era mais do que necessário desenvolver um sistema operacional para dar suporte a esta arquitetura que podia endereçar aproximadamente 4 bilhões de bytes. Mais uma vez Cutler foi escolhido para liderar este projeto de um sistema que explorasse a capacidade máxima do novo processador. Nascia, então, o VMS (Virtual Memory System), um sistema operacional de tempo compartilhado que aparecia como uma evolução do RSX-11M.

Em 1978, a Digital lançava seu primeiro computador baseado na nova arquitetura, o VAX 11/780, onde o VMS seria o sistema operacional desta plataforma e de todos os demais modelos desta família de processadores. Ao longo dos anos 80, o VMS se consolidou como um sistema operacional de grande sucesso tanto na área comercial quanto no meio acadêmico.

No início dos anos 1990, a Digital lançou o Alpha AXP, processador com arquitetura de 64 bits. Uma nova versão do VMS foi desenvolvida para suportar mais esta plataforma. A evolução do VMS permitiu que fossem incorporadas à sua arquitetura características de sistemas abertos, de acordo com padrões de interface do IEEE (Institute of Electrical and Electronics Engineers)

e especificações do OSF (Open Software Foundation) e do consórcio X/Open. A partir dessas novas funcionalidades, a Digital rebatizou o sistema como OpenVMS.

2. Características

O OpenVMS é um sistema operacional multiprogramável e multiusuário utilizado em ambientes de processamento comercial e científico que suporta diversos estilos de operação, incluindo tempo compartilhado, tempo real e até mesmo processamento de transações on-line. O sistema foi projetado para operar em plataformas VAX (CISC) e Alpha AXP (RISC) e possui como principais características:

- Implementação em ambientes cliente-servidor.
- Processamento em modo batch.
- Processamento de aplicações em tempo real.
- Interface de comandos e windows.
- Compatibilidade de códigos fonte interplataformas.
- Linguagem de controle sofisticada.
- Implementação em sistemas com multiprocessamento simétrico (SMP).
- Implementação de sistemas tolerantes à falha.
- Funcionalidades de sistemas abertos.
- Kernel do sistema operacional suportado por system services.

3. Estrutura do Sistema

A estrutura do OpenVMS consiste em quatro camadas concêntricas, variando da mais privilegiada (kernel) para a menos privilegiada (user). A seguir são apresentados os componentes do sistema associados a cada uma dessas camadas:

- Modo Kernel: subsistema de E/S, subsistema de memória, escalonamento e system services.
- Modo Executive: Record Management Services (RMS).
- Modo Supervisor: Interpretador de comandos (CLI) e run-time library.
- Modo User: utilitários e comandos DCL e programas de usuários.

4. Processo

O processo no OpenVMS pode ser dividido em quatro partes distintas: imagem, contexto de software, contexto de hardware e espaço de endereçamento virtual. Na terminologia da Digital, uma imagem é o resultado da compilação e linkedição de um programa fonte em qualquer linguagem de alto nível, ou seja, é um programa executável. Para que uma imagem possa ser executada, ela necessita de recursos do sistema, como privilégios e quotas. A totalidade desses recursos pode ser entendida como o processo. A imagem é sempre executada no contexto de um processo.

O contexto de software identifica o processo, seu dono, privilégios e quotas de recursos do sistema. Neste ambiente é determinado o que o processo pode ou não fazer, como, por exemplo, o número máximo de arquivos abertos simultaneamente.

O contexto de hardware armazena o conteúdo dos registradores gerais e do registrador de status. Com estas informações é possível que um processo tenha sua execução interrompida temporariamente, retornando posteriormente sem nenhum problema.

O espaço de endereçamento virtual consiste em uma sequência de endereços que uma imagem pode referenciar no intervalo entre 0 e 2^{32} nos processadores VAX e de 0 a 2^{64} nos Alpha AXP. O mecanismo de memória virtual implementa todo esse espaço de endereçamento utilizando a memória principal e a secundária.

Um processo pode criar outros processos, dependentes ou não de seu criador. Quando depende do seu criador, um processo é definido como subprocesso; no caso de ser independente, é chamado detached. Um subprocesso compartilha quotas com o processo-pai. Além disso, caso o processo-pai deixe de existir, o subprocesso também é eliminado. Diferentemente do subprocesso, processos detached não dependem da existência permanente do processo criador.

Durante seu ciclo de vida, um processo passa por diferentes estados no sistema. A seguir são relacionados os estados e as mudanças de estados de um processo.

- **Execução (CURrent):** o estado de execução (CUR) indica que o processo está de posse da UCP, sendo executado.
- **Pronto (COMputable):** o estado de pronto (COM) indica que o processo aguarda apenas por uma chance para ser processado, ou seja, é uma espera pelo uso da UCP. Também existe a possibilidade de o processo aguardar pela UCP fora da memória principal. Nesse caso, seu estado é dito computable outswapped (COMO).
- **Espera (Wait):** o estado de espera indica que o processo aguarda por algum evento ou recurso do sistema para continuar sua execução. No OpenVMS, os estados de espera se dividem em subestados, como LEF, CEF, HIB, SUSP, PFW, FPG, COLPG e MWAIT. Do mesmo modo que no estado de pronto, um processo pode permanecer esperando por um evento ou recurso fora da memória principal (outswapped), como LEFO, CEFO, HIBO, SUSPO, PFWO, FPGO, COLPGO e MWAITO. A tabela abaixo ilustra diferentes tipos de estado de espera.

Estado	Descrição
LEF	Espera por um local event flag.
CEF	Espera por um common event flag.
HIB	Processo em hibernação.
SUSP	Processo suspenso.
PFW	Espera devida à ocorrência de page fault.
FPG	Espera por uma página livre.
COLPG	Espera por page fault em página compartilhada.
MWAIT	Espera por mutex ou por outro recurso qualquer do sistema.

5. Gerência do Processador

A gerência do processador implementada pelo OpenVMS define a política de divisão do tempo do processador entre os processos dos usuários e do sistema operacional. O escalonamento de processos é realizado por uma rotina de interrupção do sistema denominada scheduler.

A política de escalonamento é implementada através de prioridades associadas aos processos, denominadas prioridades base. Esta prioridade é estabelecida no momento da criação do processo. O OpenVMS implementa 32 níveis de prioridades, divididos em duas faixas: tempo compartilhado (time-sharing) de 0 a 15 e tempo real (real-time) de 16 a 31.

5.1 Escalonamento de Tempo Compartilhado

Um processo em estado corrente (CUR), trabalhando na faixa de tempo compartilhado, somente deixa a UCP caso ocorra uma destas situações:

- término de execução da imagem;
- processo de maior prioridade entra em estado de COM (preempção por prioridade);
- solicitação de um evento ou recurso do sistema;
- término da fatia de tempo (prioridade por tempo).

Para o escalonamento de tempo compartilhado, além da prioridade base definida na criação do processo, existe uma outra, chamada dinâmica, que varia de acordo com as características de execução do processo. O escalonamento de tempo compartilhado é realizado com base na prioridade dinâmica dos processos.

A prioridade dinâmica é alterada quando um processo sai do estado de espera para o estado de pronto. O sistema incrementa um valor à prioridade base em função do tipo de espera a que o processo estava submetido. Eventos que exigem longo tempo de espera incidem em um incremento maior. Com isso, um processo CPU-bound tende a ter uma prioridade dinâmica menor que a de um processo I/O-bound. Este esquema permite balancear o uso do processador entre todos os tipos de processos.

A prioridade dinâmica é calculada pela soma da prioridade base com o incremento recebido. Seu valor é decrementado ao longo do tempo, porém nunca poderá cair abaixo da prioridade base estabelecida.

5.2 Escalonamento de Tempo Real

Um processo em estado corrente (CUR), trabalhando na faixa de tempo real, somente deixa a UCP caso ocorra uma das seguintes situações:

- término de execução da imagem;
- processo de maior prioridade entra em estado de COM (preempção por prioridade);
- solicitação de um evento ou recurso do sistema.

Existem duas diferenças na política de escalonamento para esses tipos de processos: a não existência do conceito de fatia de tempo e de prioridade dinâmica.

▶ 6. Gerência de Memória

A gerência de memória está intimamente ligada à arquitetura do processador. Em função disso existem pequenas diferenças no OpenVMS entre a implementação deste subsistema nos processadores VAX e Alpha AXP. Neste item, a abordagem será relativa apenas à arquitetura VAX.

A arquitetura VAX dispõe de hardware específico para a gerência da memória virtual. Este hardware possibilita que o sistema operacional ofereça uma forma flexível e eficiente na implementação da memória virtual. Podemos destacar como principais características:

- uma lógica de espaço de endereçamento virtual linear;
- mecanismo de proteção da memória;
- compartilhamento de código e dados do sistema operacional;
- mecanismos de tratamento para exceções geradas por referências a páginas não residentes.

6.1 Espaço de Endereçamento Virtual

A arquitetura VAX proporciona um espaço de endereçamento virtual para cada processo de 4 Gbytes. Grande parte desse espaço é utilizada para informações de controle, estando disponível para o programa e dados aproximadamente 1 Gbyte.

O espaço de endereçamento virtual que cada processo possui é um espaço imaginário e contíguo de células. Os endereços referenciados por um programa são sempre virtuais. Em tempo de execução, tais endereços virtuais são transformados em endereços físicos. O subsistema de gerência de memória é responsável por simular a existência do espaço virtual.

A implementação da memória virtual é realizada através de um mecanismo chamado mapeamento. Esta é a maneira pela qual o sistema consegue encontrar a correta localização física (em disco ou memória física) correspondente a um endereço virtual.

A gerência de memória implementa a técnica de paginação, responsável por manter em memória física somente a parte da memória virtual do processo em uso. Nos sistemas VAX, a página é definida como tendo 512 bytes. Com base nisso, tanto a memória virtual quanto a memória física são divididas em blocos de tamanho de 512 endereços. A memória virtual é dividida, então, em blocos de 512 endereços virtuais, denominados páginas virtuais. Da mesma forma, a memória principal é dividida em blocos de 512 células de 1 byte, denominados páginas físicas ou frames.

Fixar em 512 o tamanho da página é o resultado de uma série de ajustes. Entre estes, o fato de que a quantidade mínima de dados que o disco, nos sistemas VAX, transaciona é de 512 bytes (1 bloco). Existem também outros motivos que, por serem inerentes ao mapeamento, serão vistos no decorrer deste assunto.

O conjunto de frames que um processo possui na memória principal em determinado instante é denominado working set. Quando um programa é executado, parte de seu código e/ou dados está no working set, enquanto o restante é deixado em disco. Embora algumas páginas estejam no working set do processo e, consequentemente, na memória principal, elas ainda estão escritas em termos de endereços virtuais e, portanto, suas instruções precisam ser mapeadas antes de serem executadas pela UCP.

Para fazer o mapeamento entre a memória virtual e a memória principal, o sistema utiliza uma estrutura de dados chamada tabela de páginas (page table). Cada página da memória virtual gera um registro na tabela de páginas denominado Page Table Entry (PTE). Toda PTE tem, entre outras informações, um bit que indica se a página associada àquela PTE está ou não no working set do processo (bit de validade).

Sempre que se referencia uma página, o bit de validade é verificado. O fato de ele estar ligado indicará que a página é válida, pois está no working set. Desta forma, o sistema poderá localizar a página na memória real, através das demais informações contidas na PTE. Por outro lado, se o bit de validade não estiver ligado, significará que esta é uma página não válida, pois não está no working set. Neste caso, podemos dizer que ocorreu um page fault, que é a exceção gerada quando uma página referenciada não está no working set. Quando isto acontece, o sistema se encarregará de trazer a página para o working set, atualizar a PTE e executar novamente a referência à página, que agora será válida.

6.2 Endereço Virtual

O endereço virtual é estruturado em duas partes, correspondentes à página virtual e ao byte dentro da página. O virtual page number (VPN, bits 9:29) indica a página virtual em que o endereço está contido, e o campo de deslocamento (bits 0:8) indica qual o byte especificado dentro da página.

O espaço de endereçamento virtual de cada processo é dividido em duas regiões denominadas região do processo e região do sistema. O bit mais significativo do endereço virtual (bit 31) indica se o endereço pertence à região do processo (bit 50) ou à do sistema (bit 51).

A região do sistema é compartilhada por todos os processos. Neste caso, quando um processo faz referência a um endereço virtual dessa região, este tem o mesmo significado para todos os processos. Isto significa que quando dois processos fazem referência a um mesmo endereço virtual da região do sistema, eles estarão referenciando o mesmo código ou dado. Todo código do sistema operacional compartilhado entre processos está localizado nesta região. Para que este

compartilhamento seja realizado de forma organizada, o sistema implementa mecanismos de proteção para restringir o acesso às páginas.

Tanto a região do processo quanto a região do sistema são subdivididas em duas regiões, totalizando quatro regiões no espaço de endereçamento virtual. Cada uma dessas sub-regiões representa 1 Gbyte de endereçamento. As duas sub-regiões do espaço do processo são denominadas, respectivamente, região do programa (P0) e região de controle (P1).

Quando se executa um programa do usuário, o módulo executável é alocado em endereços virtuais da região P0, começando no endereço 0 e crescendo para os endereços maiores. Já na região P1, os endereços virtuais são alocados na sequência dos maiores para os menores, em vista de tal região ser utilizada por estruturas de pilhas.

A região do sistema também é dividida em duas sub-regiões de 1 Gbyte (S0 e S1). Os endereços virtuais da sub-região S0 contêm o código e estruturas do sistema operacional, como rotinas de interrupções, rotinas do sistema e tabelas de páginas. A sub-região S1 é reservada para uso futuro. Atualmente, a UCP não traduz um endereço virtual localizado nesta região. Caso algum processo faça referência a um endereço dentro dessa faixa, um erro de hardware ocorrerá.

6.3 Working Set

O working set de cada processo tem um tamanho máximo que limita o número de frames na memória principal que um processo pode possuir. Esta é uma forma que o sistema implementa para aumentar o grau de compartilhamento da memória entre os diversos processos. Quando este limite é alcançado e o processo necessita trazer uma nova página para a working set, um page fault acontece, e uma página deve ser cedida em troca de uma nova página.

O OpenVMS implementa a política de descartar a página mais antiga presente no working set. Para isso, o sistema mantém uma estrutura chamada lista do working set (working set list) para cada processo. Aparentemente esta política pode não parecer tão boa, se partirmos do princípio de que a página descartada pode ser uma página muito referenciada. Posteriormente entenderemos que o sistema implementa mecanismos que compensam a opção por essa política.

No momento em que uma página é retirada do working set, esta pode tomar rumos diferentes em função das suas características. Para entender como isso ocorre, precisamos antes distinguir as páginas modificáveis das não modificáveis. As páginas modificáveis são aquelas que contêm dados ou variáveis que naturalmente trocam de valor durante a execução do programa. Por outro lado, as páginas que contêm código não podem ser alteradas, sendo por isso chamadas de não modificáveis. Uma página modificada não pode ser jogada fora quando sai do working set, pois ela contém dados necessários à continuidade da execução do programa. É preciso, então, que essa página, ao sair do working set, seja gravada em disco para que, na próxima referência, a mesma volte com os valores preservados. As páginas não modificadas, por sua vez, não precisam desse tratamento, pois podem ser retiradas do arquivo executável quando necessário.

O tempo de leitura e o de gravação em disco são reconhecidamente grandes quando comparados com o tempo de acesso à memória. Imagine se a cada entrada ou saída de página do working set fossem feitas leitura e gravação em disco. Para contornar esse problema o sistema mantém duas listas que funcionam em conjunto no sentido de diminuir as operações com disco. Essas listas são a lista de páginas livres (free page list) e a lista de páginas modificadas (modified page list), que funcionam como será descrito.

Além do bit de validade, cada PTE tem um bit de modificação que indica se a página sofreu alteração no seu conteúdo ou não. Assim, quando uma página é retirada do working set para que outra entre, o bit de modificação é verificado. Caso este esteja ligado indicando que houve modificação, a página é passada da lista do working set do processo para a lista de páginas modificadas. Por outro lado, se o bit estiver desligado, a página é passada para a lista de páginas livres.

A lista de páginas livres é responsável por manter informações sobre as páginas físicas de memória que estão disponíveis para uso. Quando uma página é liberada do working set do processo para a lista de páginas livres, esta página não é imediatamente utilizada, já que a página deve percorrer a lista até seu início. Desta forma, as páginas que saem do working set permanecem na memória por algum tempo, permitindo que essas mesmas páginas possam voltar ao working set sem idas ao disco.

Na ocorrência de um page fault, o sistema pode eventualmente encontrar a página requerida na lista de páginas livres ou na lista de páginas modificadas, gerando assim o que é chamado de page fault barato (page fault sem operação de E/S). Quando não ocorre o page fault barato, o sistema é obrigado a trazer a página requerida do disco, utilizando um novo frame na qual a página será colocada.

A lista de páginas livres fica, eventualmente, com poucas páginas disponíveis para atender a demanda dos processos. Nesse caso, as páginas que estão na lista de páginas modificadas são gravadas em disco (arquivo de paginação) e passadas para a lista de páginas livres. Note que apenas nesse caso as páginas modificadas são gravadas, em conjunto, no disco e, portanto, apenas uma ida ao disco é efetuada.

Quando ocorre um page fault, a página requerida pode estar em vários lugares, como no arquivo executável, na lista de páginas modificadas, na lista de páginas livres ou no arquivo de paginação. O arquivo de paginação é uma área em disco comum a todos os processos, onde as páginas modificáveis são salvas para posterior reutilização.

6.4 Swapping

A paginação é um procedimento natural do mecanismo de memória virtual. Os processos estão permanentemente paginando e, para isso, o OpenVMS deve possuir sempre páginas disponíveis na memória principal para serem utilizadas. Em função disso, a lista de páginas livres deve sempre possuir uma quantidade mínima de páginas necessária para a utilização dos processos.

Em determinadas situações, a memória principal pode estar sendo bastante utilizada, levando a lista de páginas livres a valores baixos. Nesta situação específica, o sistema seleciona um ou mais processos e retira temporariamente o(s) working set(s) da memória principal, gravando-o(s) em disco (arquivo de swap). Com isso, páginas da memória principal são desocupadas, permitindo que o mecanismo da paginação continue seu funcionamento.

▶ 7. Sistema de Arquivos

O OpenVMS trabalha com arquivos em um formato próprio, conhecido como Files-11 ODS-2 (On-Disk Structure Level 2). Essa arquitetura de disco define como os dados são armazenados em disco e os arquivos de controle necessários para dar suporte a essa arquitetura.

A menor estrutura lógica endereçável em um disco Files-11 ODS-2 é um bloco, composto por 512 bytes. A alocação de espaço em disco, para a criação e extensão de arquivos, é feita utilizando uma unidade denominada cluster. Um cluster é formado por 1 ou mais blocos contíguos, podendo variar entre 1 e 65.535. O tamanho do cluster é definido durante a inicialização do disco. Um conjunto de clusters contíguos é chamado de extent, podendo um arquivo ser formado por um ou mais extents.

O sistema de arquivos do OpenVMS tem como principais características:

- identificação de arquivos com até 39 caracteres para o nome e mais 39 caracteres para extensão;

- estrutura de diretório em árvore de até oito níveis;
- organização de arquivos sequencial, relativa e indexada;
- rotinas RMS para operações de E/S;
- proteção por grupo e por lista de controle de acesso;
- alocação de espaço em disco não contígua;
- detecção automática de blocos com defeitos;
- controle de alocação de espaço em disco por usuário.

8. Gerência de Entrada/Saída

A gerência de entrada/saída do OpenVMS é estruturada em camadas, em que as camadas superiores escondem os detalhes das camadas inferiores.

A camada de mais alto nível é o RMS (Record Management Services), que dá suporte a acesso a arquivos e registros. Todas as linguagens de alto nível fazem uso do RMS para executar operações de E/S.

A camada intermediária, composta pelas system services de E/S, é responsável pela interface entre o código do sistema operacional dependente do dispositivo e as aplicações do usuário. As chamadas às system services $QIO permitem executar operações de E/S não suportadas pelo RMS.

O device driver é o código do sistema operacional com características específicas de cada dispositivo conectado ao sistema. Uma aplicação de usuário raramente trabalha neste nível para operações de E/S.