APOSTILA DE SISTEMAS OPERACIONAIS

PROF. OSCAR SANTANA

CURSO TÉCNICO EM INFORMÁTICA

ADVICE

1- Introdução a sistemas operacionais

1.1 – O Que É Um Sistema Operacional

Para que o hardware ou parte física de um computador possa funcionar, faz-se necessário um conjunto de regras e ordens que coordenem todos os processos realizados. Tal conjunto é denominado software ou parte não material do sistema. Graças ao software (integrado por uma enorme quantidade de programas que interagem entre si) todos os recursos podem ser utilizados em qualquer sistema informatizado.

Todo o conjunto de programas que compõem o software pode ser dividido em dois grupos bem diferenciados:

1. Software básico: conjunto de programas imprescindíveis para o funcionamento do sistema. (Drivers controladores de hardware)
2. Software aplicativo. Conjunto de programas a serem utilizados pelo usuário(Word, Internet Explorer, Paciência).

À esse software, dá-se o nome de Sistema Operacional.

O Sistema Operacional

O sistema operacional tem duas funções distintas: estender a máquina e gerenciar recursos.

Como máquina estendida, o sistema operacional oculta a ‘verdade’ do usuário sobre o hardware e apresenta uma visão simples e agradável. Ele evita, por exemplo, que o usuário tenha que gerenciar o HD para gravar dados, e apresenta uma interface orientada a arquivos simples, geralmente em estrutura de pastas e diretórios. O sistema operacional também é responsável por fornecer uma variedade de serviços que os programas podem obter usando instruções especiais conhecidas como chamadas ao sistema, isso sem que o usuário tenha que interagir diretamente com a máquina.

Como gerenciador de recursos, o sistema operacional controla de forma ordenada o uso dos dispositivos físicos entre os vários programas que competem por eles. Esse gerenciamento é feito através de compartilhamento no tempo e no espaço. Quando um dispositivo é compartilhado no tempo, cada programa ou usuário aguarda a sua vez de usar o recurso(Processador). Quando um dispositivo é compartilhado no espaço, cada programa ou usuário ocupa uma parte do recurso (Memória RAM).

A diversidade de sistemas operacionais

No topo da lista estão os sistemas operacionais para computadores de grande porte. Esses computadores exigem grande capacidade de recursos de entrada/saída de dados. Seus sistemas operacionais são orientados para o processamento simultâneo de muitos trabalhos (jobs). Eles oferecem normalmente três tipos de serviços: em lote, processamento de transações e tempo compartilhado. Um exemplo é o OS/360.

Um nível abaixo estão os sistemas operacionais de servidores. Eles são executados em servidores, em estações de trabalho ou em computadores de grande porte. Eles servem múltiplos usuários de uma vez em uma rede e permitem-lhes compartilhar recursos de hardware e software. Exemplos incluem Linux e Windows 2003 Sever.

Há também os sistemas operacionais de multiprocessadores. Esse sistema consiste em conectar várias CPUs em um único sistema para ganhar potência computacional. Eles usam variações dos sistemas operacionais de servidores com aspectos especiais de comunicação e conectividade.

O nível seguinte é o sistema de computadores pessoais. Sua função é oferecer uma boa interface para um único usuário. Exemplos comuns são o Windows Vista e XP, o MacOS e o Linux.

O próximo nível é o de sistemas de tempo real. O tempo é um parâmetro fundamental. Eles são divididos em sistemas de tempo real crítico e de tempo real não crítico. Os sistemas de tempo real crítico possuem determinados instantes em que as ações devem ocorrer. Os sistemas de tempo real não crítico aceita um descumprimento ocasional de um prazo. VxWorks e QNX são exemplos bem conhecidos.

Descendo na escala, vemos os sistemas operacionais móveis e embarcados. Os sistemas móveis estão presentes em computadores de mão que são computadores muito pequenos que realizam funções de agenda e livro de endereços. Os sistemas embarcados são computadores que controlam eletrodomésticos ou sistemas de comunicação e de orientação por gps em veículos. Exemplos de sistemas operacionais móveis são o PalmOS e o Windows Mobile. De sistema operacional embarcado pode-se citar o Microsoft SYNC e o sistema de GPS do Fiat Linea.

Os menores sistemas operacionais são executados em cartões inteligentes. São dispositivos do tamanho de cartões de crédito que contêm uma CPU. Possuem restrições severas de consumo de energia e memória. Alguns são orientados a Java. Um exemplo é o SmartCard da Athos Sistemas do Brasil, utilizado em sistemas de controle de acesso e gerenciamento de estoques.

Exemplos de sistemas operacionais comuns.

● Microsoft Windows Vista e XP – Sistema operacional para estações de trabalho, sejam elas domésticas ou em ambiente corporativo.
● Microsoft Windows Server 2003 – Sistema operacional para servidores, com recursos para gerenciamento de usuários e estações de trabalho.
● Linux (CentOS, Fedora, Debian, Suse, Slackware, Kurumin)– Sistema operacional utilizado tanto em servidores quanto estações de trabalho. Possui uma ampla disponibilidade de aplicativos compatíveis, como servidores de rede, aplicativos multimídia, entre outros.

1.2 A história dos sistemas operacionais.

Os sistemas operacionais têm sido historicamente amarrados à arquitetura dos computadores nos quais iriam rodar. Por isso, veremos como eles evoluiram nas sucessivas gerações de computadores. Esse mapeamento entre gerações de computadores e gerações de sistemas operacionais é admissivelmente imaturo, mas tem algum sentido.

O primeiro computador digital verdadeiro foi projetado pelo matemático inglês Charles Babbage (1792-1871). Embora Babbage tenha dispendido muito de sua vida e de sua fortuna tentando construir sua "máquina analítica", ele jamais conseguiu por o seu projeto em funcionamento porque era simplesmente um modelo matemático e a tecnologia da época não era capaz de produzir rodas, engrenagens, dentes e outras partes mecânicas para a alta precisão que necessitava. Desnecessário se dizer que a máquina analítica não teve um sistema operacional.

1 - A Primeira Geração (1945-1955): Válvulas e Painéis com Plugs

Após os esforços sem sucesso de Babbage, pouco progresso se teve na construção de computadores digitais até a Segunda Guerra Mundial. Em torno de 1940, Howard Aiken em Harvard, John Von Neumann no Instituto para Estudos Avançados em Princeton, John Presper Eckert e William Mauchley na Universidade de Pennsylvania e Konrad Zuse na Alemanha, entre outros, tiveram sucesso na construção de máquinas calculadoras usando válvulas. Essas máquinas eram enormes, ocupando salas completas, com dezenas de milhares de válvulas, porém eram muito mais lentas do que os mais simples computadores pessoais de hoje.

Naqueles dias primitivos, um pequeno grupo de pessoas construiu, programou, operou e deu manutenção a cada máquina. Toda a programação era feita em linguagem de máquina, sempre se conectando fios com plugs em painéis para controlar as funções básicas da máquina. As linguagens de programação não eram conhecidas (nem a linguagem Assembly). Nem se ouvia falar em sistemas operacionais. O modo usual de operação consistia no programador elaborar o programa numa folha e então ir à sala da máquina, inserir os plugs nos painéis do computador e gastar as próximas horas apelando para que nenhuma das 20.000 ou mais válvulas se queimasse durante a execução do programa. Na verdade, todos os problemas eram inerentemente sobre cálculos numéricos tais como gerações de tabelas de senos e cossenos.

Por volta dos anos 50, essa rotina teve uma pequena evolução com a introdução de cartões perfurados. Era possível, a partir de então, se escrever programas em cartões e lê-los, em vez do uso de plugs em painéis; no mais, o procedimento era o mesmo.

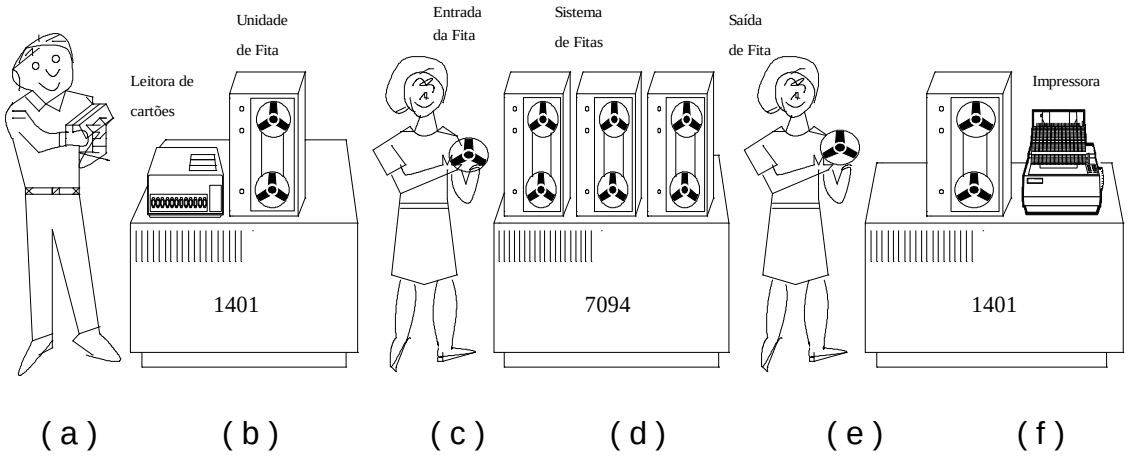
2 - A Segunda Geração (1955 - 1965): Transistores e Sistemas Batch

A introdução do transistor em meados dos anos 50 mudou o quadro radicalmente. Os computadores tornaram-se bastante confiáveis para que pudessem ser produzidos e vendidos comercialmente na expectativa de que eles continuassem a funcionar por bastante tempo para realizar algumas tarefas usuais. A princípio havia uma clara separação entre projetistas, construtores, operadores, programadores e o pessoal de manutenção.

Essas máquinas eram alocadas em salas especialmente preparadas com refrigeração e com apoio de operadores profissionais. Apenas grandes companhias, agências governamentais, ou universidades, dispunham de condições para pagar um preço de milhões de dólares por essas máquinas. Para rodar um job (isto é, um programa ou um conjunto de programas), primeiro o programador escrevia o programa no papel (em FORTRAN ou linguagem Assembly), e então perfurava-o em cartões. Daí, ele levava o conjunto de cartões chamado de "deck", à sala de recepção e o entregava a um dos operadores.

Quando o computador encerrava a execução de um job, um operador apanhava a saída na impressora, a conduzia de volta à sala de recepção onde o programador poderia coletá-lo posteriormente. Então ele tomava um dos decks de cartões que tinha sido trazido da sala de recepção e produzia a sua leitura. Se o compilador FORTRAN era necessário, o operador tinha que pegá-lo de uma sala de arquivos e produzir a sua leitura. Muito tempo de computador era desperdiçado enquanto os operadores caminhavam pela sala da máquina para realizarem essas tarefas.

Devido ao alto custo do equipamento, era de se esperar que as pessoas tentassem reduzir o tempo desperdiçado. A solução geralmente adotada era o sistema em "batch". A idéia original era colecionar uma bandeja completa de jobs na sala de recepção e então lê-los para uma fita magnética usando um computador pequeno e relativamente barato, por exemplo o IBM 1401, que era muito bom na leitura de cartões, na cópia de fitas e na impressão da saída, porém não era tão bom em cálculo numérico. Outros computadores, máquinas mais caras, tais como o IBM 7094, eram usados para a computação real. Essa situação é mostrada na figura 1.



(a) (b) (c) (d) (e) (f)

Figura 1 - Um sistema “batch” antigo.
(a) Programadores levam cartões ao 1401.
(b) 1401 grava batch de jobs em fita.
(c) A operadora acopla fita de entrada no 7094.
(d) O 7094 faz o processamento.
(e) A operadora acopla fita de saída no 1401.
(f) O 1401 imprime a saída.

Após cerca de uma hora coletando-se um lote de jobs, a fita era rebobinada e levada para a sala da máquina onde era montada numa unidade de fita. O operador então carregava um programa especial (o antecessor do sistema operacional de hoje), que lia o primeiro job da fita e o executava. A saída era escrita numa segunda fita, em vez de ser impressa. Após o fim da execução de cada job, o sistema operacional automaticamente lia o próximo job da fita e começava a executá-lo. Quando todo o "batch" era feito, o operador removia as fitas de entrada e de saída, substituia a fita de entrada pelo próximo "batch" e levava a fita de saída para um 1401 produzir a impressão "off-line" (isto é, não conectada ao computador principal).

A estrutura de um job de entrada típico é mostrada na figura 2. Ele começa com um cartão $JOB, especificando o tempo máximo de execução em minutos, o número da conta e o nome do programador. A seguir vinha um cartão $FORTRAN, avisando ao sistema operacional para carregar o compilador FORTRAN da fita do sistema. Em seguida vinha um programa a ser compilado, acompanhado de um cartão $LOAD, informando ao sistema operacional para carregar o programa objeto já compilado. (Programas compilados eram sempre escritos em fitas selecionadas e tinham de ser carregadas explicitamente). A seguir vinha um cartão $RUN, informando ao sistema operacional para executar o programa com os dados que vinham a seguir. Finalmente o cartão $END marcava o fim do job. Esses cartões de controle foram os precurssores das linguagens de controle de job (JCL) modernas e de interpretadores de comandos.

Muitos computadores da segunda geração foram usados principalmente para cálculos científicos e de engenharia, tais como em solução de equações diferenciais parciais. Eles foram vastamente programados em FORTRAN e em

linguagem Assembly. Sistemas operacionais típicos eram o FMS (Sistema Monitor FORTRAN) e IBSYS (Sistema Operacional da IBM para o 7094).

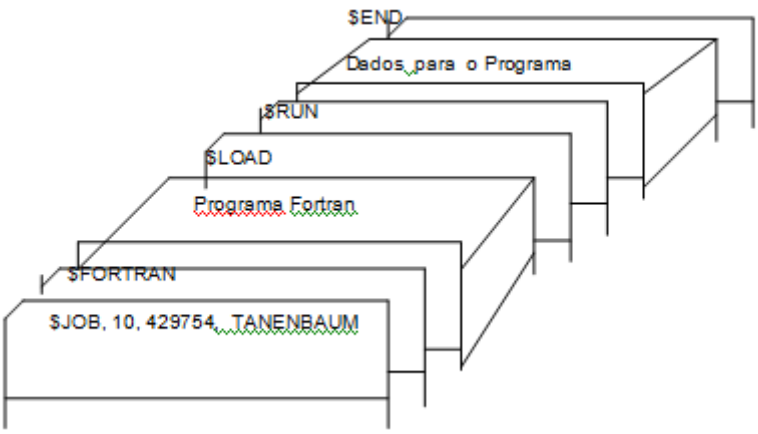


Figura 2 - Estrutura de um típico job FMS

3 - A Terceira Geração (1965 - 1980): CIs e Multiprogramação

Nos anos 60, muitos fabricantes de computadores tinham duas linhas de produto distintas e totalmente incompatíveis. Por um lado havia os computadores científicos, em grande escala, orientado por palavras, tais como o 7094, que era usado para cálculos numéricos em ciência e engenharia. Por outro lado, havia os computadores comerciais, orientados por caracter, tais como o 1401, que era vastamente usado para classificação em fita e impressão, por bancos e companhias de seguros.

O desenvolvimento e a manutenção de duas linhas de produto completamente diferentes era uma proposta cara para os fabricantes. Além do mais, os clientes em potencial para aquisição de novos computadores necessitavam inicialmente de uma máquina pequena, para mais tarde, com o crescimento, terem uma máquina maior em que pudessem rodar todos os seus programas mais rapidamente.

A IBM, no intuito de resolver ambos os problemas de uma só tacada, introduziu o sistema /360. O 360 era uma série de máquinas compatíveis por software, variando de tamanho a partir do 1401 até o mais potente 7094. As máquinas diferiam apenas em preço e performance (capacidade de memória, velocidade do processador, número de periféricos I/O permitidos, e assim por diante). Já que todas as máquinas tinham a mesma arquitetura e o mesmo conjunto de instruções, pelo menos em teoria, programas escritos para uma máquina poderiam rodar em todas as outras. Além disso, o 360 foi projetado para manusear tanto computação comercial como computação científica. Assim, uma única família de máquinas poderia satisfazer às necessidades de todos os clientes. Em anos subsequentes, a IBM apresentou os sucessores compatíveis com a linha 360, usando uma tecnologia mais moderna, conhecidos como séries 370, 4300, 3080 e 3090.

O 360 foi a primeira linha de computadores a usar (em pequena escala) circuitos integrados (CIs), fornecendo uma maior vantagem em

preço/performance sobre as máquinas da segunda geração, que eram construidas de transistores individuais. Isso foi um sucesso imediato e a idéia de uma família de computadores compatíveis foi logo adotada por todos os outros fabricantes. Os descendentes dessas máquinas ainda hoje estão em uso em grandes centros de computação.

A maior força da idéia de "uma família" foi simultaneamente a sua maior durabilidade. A intenção era que todo o software, incluindo o sistema operacional, deveria trabalhar em todos os modelos. Ele tinha de rodar em sistemas pequenos que muitas vezes já substituia 1401s para cópias de cartões em fitas e em sistemas muito grandes, que muitas vezes substituia 7094s para fazer cálculos demorados e outras computações pesadas. Ele deveria ser bom em sistemas com poucos periféricos e em sistemas com muitos periféricos. Ele tinha de trabalhar em ambientes comerciais e em ambientes científicos. Acima de tudo, ele tinha de ser eficiente em todos esses usos diferentes.

Não havia uma maneira através da qual a IBM (ou outra companhia) pudesse solucionar todas essas exigências conflitantes. O resultado foi um sistema operacional enorme e extraordinariamente complexo, provavelmente de dois ou três ordens de magnitude maior do que o FMS. Ele consistia de milhares de linhas de linguagem assembly escritas por centenas de programadores e continha centenas e centenas de depurações que necessitavam de contínuas versões a fim de corrigí-los. Cada nova versão fixava algumas depurações e introduzia outras novas, tal que o número de depurações provavelmente permanecia constante com o tempo.

A despeito de seu enorme tamanho e de seus problemas, o OS/360 e os sistemas operacionais similares da terceira geração, produzidos por outros fabricantes, satisfizeram razoavelmente bem a seus clientes. Eles também popularizaram várias técnicas ausentes nos sistemas operacionais da segunda geração. Provavelmente, a mais importante dessas técnicas foi a multiprogramação. No 7094, quando o job que estava sendo executado tinha uma pausa esperando que uma operação em fita ou em qualquer outro periférico I/O fosse completada, a CPU simplesmente ficava ociosa até que a operação I/O fosse encerrada. Em cálculos científicos pesados, as operações de I/O não são frequentes, e essse tempo ocioso é insignificante. Em processamento de dados comerciais, as operações de I/O consomem frequentemente entre 80 a 90 porcento do tempo total, exigindo alguma providência sobre isso.

A solução foi particionar a memória em várias partes, com um job diferente em cada partição, como mostrado na Fig. 3. Enquanto um job estava esperando que uma operação I/O fosse concluida, um outro job poderia usar a CPU. Se vários jobs pudessem ocupar a memória no mesmo instante, a CPU estaria sempre ocupada quase que em 100% do tempo. Ter múltiplos jobs na memória, por sua vez, requer hardware especial para proteger cada job contra danos e entrelaçamento entre eles, e o 360 e outros sistemas da terceira geração eram equipados com esse hardware.

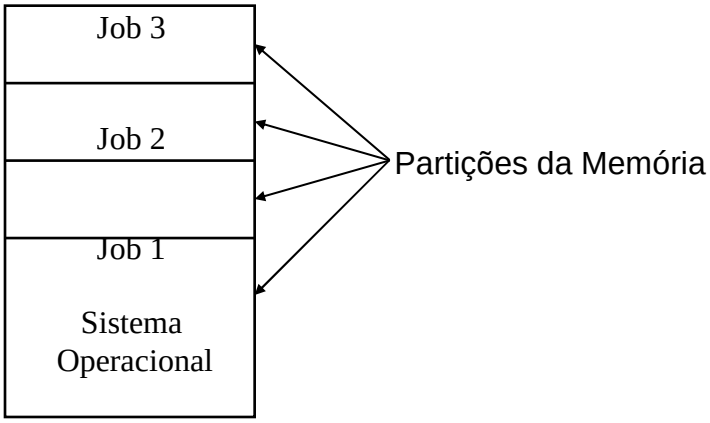


Figura 3 - Um sistema de multiprogramação com três jobs na memória

Um outro grande aspecto presente nos sistemas operacionais da terceira geração era a habilidade de ler jobs de cartões para o disco assim que eles eram trazidos à sala do computador. Assim, sempre que um job tinha a sua execução encerrada, o sistema operacional poderia carregar um novo job do disco numa nova partição vazia e executá-lo. Essa técnica é chamada de "spooling" (de "Simultaneous Perifheral Operation On Line") e também era usada para a saída. Com o "spooling", os 1401s não precisavam ser tão grandes e a utilização da fita diminuiu bastante.

Apesar dos sistemas operacionais da terceira geração terem sido bem apropriados para a execução de programas envolvendo grandes cálculos científicos e de processamento de dados comerciais compactos, eles eram ainda, basicamente, sistemas em "batch". Muitos programadores sentiam saudades dos dias da primeira geração, quando eles tinham a máquina toda para eles por poucas horas, mas de tal forma que eles depuravam os seus programas rapidamente. Com os sistemas da terceira geração, o tempo entre a submissão do job e a obtenção da saída era frequentemente de várias horas, a ponto da ausência de uma única vírgula causar uma falha na compilação e o programador desperdiçava quase um dia.

A vontade de ter um tempo de resposta menor abriu espaço para "time-sharing", uma variante da multiprogramação, em que cada usuário tem um terminal "on-line". Num sistema "time-sharing", se 20 usuários estão conectados e 17 deles estão pensando, falando ou tomando café, a CPU pode ser alocada para os três jobs que querem serviço. Como as pessoas que depuram programas usualmente editam poucos comandos (como compilar um programa de cinco páginas) em vez de programas longos (como classificar mil registros em fita), o computador pode fornecer mais rápido, serviço interativo a um número maior de usuários e talvez também trabalhar com grandes jobs em "batch" paralelamente, enquanto a CPU está, por outro lado, ociosa. Embora a primeira série de sistemas em time-sharing (CTSS) foi desenvolvido no MIT num IBM 7094 especialmente modificado, ele não se tornou verdadeiramente popular até que a necessidade de proteção de hardware ficasse mais difundida durante a terceira geração.

Após o sucesso do sistema CTSS, o MIT, o Laboratório Bell e a General Electric (então o maior fabricante de computadores) decidiram embarcar no desenvolvimento de um "computador utilitário", uma máquina que suportasse milhares de usuários em "time-sharing" simultaneamente. O seu modelo era baseado no sistema de distribuição de eletricidade - quando voce precisa de eletricidade, basta por um plug na tomada da parede e a quantidade que voce precise, terá. Os projetistas desse sistema, conhecido como MULTICS (MULTiplexed Information and Computing Service), tinham em mente uma grande máquina que fornecesse serviço de computação para todos em Boston. A idéia de que máquinas tão poderosas quanto o GE44 seriam vendidas como computadores pessoais por alguns milhares de dólares apenas vinte anos mais tarde era, naquela época, pura ficção científica. Para resumir, o MULTICS introduziu muitas idéias inovadoras na literatura da computação, mas a sua construção foi mais difícil do que se esperava. O Laboratório Bell saiu do projeto e a General Electric continuou sozinha. Eventualmente o MULTICS rodava o suficientemente bem para ser usado num ambiente de produção no MIT e em poucas outros lugares, mas a idéia de um computador utilitário falhou. Mesmo assim, o MULTICS teve uma enorme influência nos sistemas subsequentes.

Outro importante desenvolvimento durante a terceira geração foi o crescimento fenomenal de mini-computadores, começando com o DEC PDP-1 em 1961. O PDP-1 tinha apenas 4 K palavras de 18 bits mas a um custo de 120.000 dólares por máquina (menos que 5% do preço de um 7094) eles vendiam como bolinhos. Para certos tipos de trabalhos não-numéricos era quase tão rápido quanto o 7094 e fez surgir uma nova indústria. Foi rapidamente seguido por uma série de outros PDPs (que diferentes da família IBM, eram todos incompatíveis) culminando com o PDP-11.

Um dos cientistas do Laboratório Bell que trabalhou no MULTICS, Ken Thompson, logo depois encontrou um pequeno PDP-7 que ninguém usava e começou a escrever uma versão simplificada mono-usuário do MULTICS. Brian Kernighan apelidou esse sistema de UNICS (UNiplexed Information and Computing Service), mas sua grafia foi mais tarde trocada para UNIX. Posteriormente foi levado para um PDP-11/20, onde funcionou bem o bastante para convencer a gerência do Laboratório Bell em investir no PDP-11/45 para continuar o trabalho.

Outro cientista do Laboratório Bell, Dennis Ritchie, juntou-se a Thompson para reescrever o sistema numa linguagem de alto nível chamada C, projetada e implementada por Ritchie. O Laboratorio Bell licensiava o UNIX para Universidades quase de graça e dentro de poucos anos, centenas delas estavam usando-o. O UNIX logo estendeu-se para o Interdata 7/32, para o VAX, para o MOTOROLA 68000, e para

muitos outros computadores. O UNIX tinha sido transportado para mais computadores do que qualquer outro sistema operacional da história e seu uso está ainda aumentando rapidamente.

4 - A Quarta Geração (1980-1990): Computadores Pessoais

Com o desenvolvimento de circuitos LSI (Large Scale Integration), chips contendo milhares de transistores em um centimetro quadrado de silício, a era do computador pessoal começava. Em termos de arquitetura, os computadores pessoais não eram diferentes de minicomputadores da classe do PDP-11, mas em termos de preço eles eram certamente bem diferentes. Enquanto o minicomputador tornou possível um departamento de uma companhia ou uma universidade ter o seu próprio computador, o chip micropocessador tornou possível um indivíduo ter o seu próprio computador.

A grande variedade de capacidade computacional disponível, especialmente a capacidade de computação altamente interativa com excelentes facilidades gráficas, fizeram crescer a indústria de produção de software para computadores pessoais. Muitos desses softwares eram "amigáveis ao usuário", significando que eles foram projetados para usuários que não tinham conhecimento algum sobre computadores e além do mais não tinha outra intenção a não ser a de orientá-los no uso. Essa foi certamente a maior mudança do OS/360, cujo JCL era tão complexo que livros inteiros foram escritos sobre ele.

Dois sistemas operacionais dominaram a utilização do computador pessoal: o MS-DOS, escrito pela Microsoft para o IBM PC e para outras máquinas que usavam a CPU Intel 8088 e seus sucessores, e UNIX, que é predominante em máquinas que usam a CPU da família Motorola 68000. Pode parecer irônico que o descendente direto do MULTICS, projetado para o gigante computador utilitário, ficou tão popular em computadores pessoais, mas principalmente mostra como foram boas as idéias sobre o MULTICS e o UNIX. Apesar da primeira versão do MS-DOS ser primitiva, em versões subsequentes foram incluidas diversas facilidades do UNIX, o que não é tão surpreendente já que a Microsoft é um dos maiores fornecedores do UNIX, usando o nome comercial XENIX.

Um interessante desenvolvimento que começou em meados dos anos 80 foi o crescimento de redes de computadores pessoais rodando sistemas operacionais para rede e sistemas operacionais distribuidos. Num sistema operacional para rede, os usuários têm consciência da existência de múltiplos computadores e podem se conectar com máquinas remotas e copiar arquivos de uma máquina para outra. Cada máquina roda o seu próprio sistema operacional local e tem o seu próprio usuário (ou usuários).

Um sistema operacional distribuido, em contraste, aparece para o usuário como um sistema tradicional de um único processador, mesmo sendo composto realmente de múltiplos processadores. Num verdadeiro sistema distribuido, os usuários não têm consciência de onde os seus programas estão sendo rodados ou onde seus arquivos estão localizados; tudo é manuseado automática e eficientemente pelo sistema operacional.

Os sistemas operacionais em rede não são fundamentalmente diferentes dos sistemas operacionais de um único processador. Eles obviamente necessitam de um controlador de interface de rede e de algum software de alto nível para gerenciá-lo, bem como de programas para concluir com êxito uma conexão remota e o acesso a arquivos remotos, mas essas adições não mudam a estrutura essencial do sistema operacional.

Os sistemas operacionais distribuidos requerem mais do que a adição de códigos a um sistema operacional de um processador porque sistemas distribuidos e centralizados diferem em modos críticos. Sistemas distribuidos, por exemplo, frequentemente admitem rodar programas em vários processadores ao mesmo tempo, e daí exigem algorítmos de escalonamento de processadores para otimimizar a quantidade de paralelismo que deve ser concluído com êxito.

O atraso de comunicação em uma rede frequentemente significa que esses (e outros) algorítmos devem rodar com informação incompleta, desatualizada ou às vezes incorreta. Essa situação é radicalmente diferente de um sistema de um único processador no qual o sistema operacional tem a informação completa sobre o estado do sistema.

Tolerância a falhas é uma outra área em que os sistemas distribuidos são diferentes. É comum para um sistema distribuido ser projetado com a expectativa de que continuará rodando mesmo que parte do hardware deixe de funcionar. Desnecessário se dizer que uma exigência adicional ao projeto tem enormes implicações para o sistema operacional.

1990/97: a Era Windows

Se o mundo da computação estava procurando por um novo padrão ou não, ele encontrou um em maio de 1990, quando a Microsoft finalmente lançou o Windows 3.0.

O Windows 3.0 era executado sobre o DOS e, portanto, oferecia compatibilidade com os programas DOS. Ele se beneficiava do processador 386, podendo fazer a multitarefa com programas DOS e também com programas Windows. A interface com o usuário foi projetada para se parecer com o Presentation Manager, trazendo um Gerenciador de Programas baseado em ícones e um Gerenciador de Arquivos em estilo árvore, incluindo avanços como ícones sombreados. Embora o Windows 3.0 tenha exigido revisões mínimas de praticamente todos os programas Windows existentes na época, não havia muito a ser revisado. Além do mais, imediatamente após a introdução do Windows 3.0, começaram a aparecer os aplicativos, liderados pela divisão de aplicativos da própria Microsoft e seguidos por praticamente todos os outros grandes desenvolvedores. Mesmo depois do anúncio do Windows 3.0, a Microsoft e a IBM continuavam falando sobre o OS/2 e, especialmente, sobre o OS/2 2.0, a primeira versão 32 bits real que viria a aparecer, finalmente, em 1992.

Para contundir ainda mais as coisas, enquanto a IBM posicionava o OS/2 como o futuro sistema operacional para todos os usuários, a Microsoft posicionava o OS/2 como um topo de linha, apenas para os aplicativos missão crítica e baseados em servidor. Em vez disto, a Microsoft começou a falar sobre o OS/2 3.0 (não confundir com o posterior IBM OS/2 Warp 3.0), que adicionaria segurança e suporte avançados a multiprocessador, sendo capaz de executar aplicativos Windows e Posix diretamente. Neste cenário, o Windows NT era o núcleo sobre o qual se apoiariam o DOS, o Windows, o OS/2 e o Posix.

As duas companhias finalmente separaram suas estratégias no início de 1991, com
Jim Cannavino, da IBM, e Bill Gates, da Microsoft, brigando como um casal durante um divórcio litigioso. O OS/2 conquistou um forte nicho em algumas grandes aplicações corporativas, auxiliado por sua estabilidade e robustez, comparadas ao Windows 3.x. Mais tarde, a IBM faria uma última tentativa de fazer do OS/2 o principal sistema operacional com seu OS/2 Warp 3.0, mais orientado ao consumidor comum e lançado no final de 1994. Ele venderia milhões de cópias mas não diminuiria a grande inclinação da indústria pelo Windows.

A Microsoft viria a transformar seu antigo "OS/2 3.0" no Windows NT 3. 1, que foi lançado em 1993 sem o suporte gráfico ao OS/2 e recebido, inicialmente, como um sistema operacional para servidores de aplicativos, concorrendo, principalmente, com o OS/2 da IBM.

Para a maioria dos usuários de PCs, a Microsoft ofereceu o Wlndows 3.1 avançado no final de 1991, que adicionava uma melhor integração de aplicativos, recursos arrastar-e-soltar e uma maior estabilidade. No início dos anos 90, ele se tornou o padrão dominante para os aplicativos para PC e a Microsoft ocupou o papel de líder na definição das especificações multimídia.

A Microsoft viria a dominar muitas áreas mais na computação por esta mesma época. Seus produtos Visual Basic e Visual C++ venceram a grande concorrência da Borland no domínio de linguagens de programação. Além disto, os aplicativos Microsoft - liderados pelo pacote Office, contendo o Word, o Excel, o PowerPoint e, mais tarde, o Access tomaram grande parte do mercado de programas aplicativos (o que foi auxiliado, em parte, pelos atrasos nas versões do Lotus 1-2-3, WordPerfect e DBASE para Windows, sendo que este último foi adquirido pela Borland).

Neste período, o Macintosh, da Apple, continuava a crescer e expandir-se e encontrou nichos nas artes gráficas, na multimídia e na educação. Mas, na maioria das empresas e órgãos governamentais, o principal sistema comercial era aquele que seguia os padrões do PC original. Àquela época, o termo *compatível com IBM já* tinha saído de moda, para ser substituído pelo Processador como a principal peça descritiva de hardware.

A era do 286 já havia terminado no final de 1988, após a introdução do 386SX da Intel, um Processador que possuía os componentes internos de

32 bits do 386 e um barramento de dados 16 bits como o 286, o que o tornava barato. Este e o 386 original rebatizado como 386DX dominaram as vendas de computadores durante anos. Em abril de 1989, a Intel apareceu com seus processadores 486. Com 1,2 milhões de transistores, o 486 era, efetivamente, uma versão mais rápida e mais refinada do 386 somada a um co-processador matemático que executava todos os aplicativos escritos para o 386 sem quaisquer problemas.

Desta vez, ninguém esperou pela IBM ou pela Compaq. Dezenas de desenvolvedores se apressaram para tornar disponíveis suas máquinas 486 o mais rápido possível após a introdução da Intel, e estas máquinas tinham uma velocidade de execução 50 vezes maior que o IBM PC original.

A Intel introduziu seu Processador Pentium de 60 MHz em março de 1993, mas não eram apenas os processadores que continuavam a avançar. Os discos rígidos ficavam cada vez maiores e mais velozes. E a tecnologia de exibição gráfica progrediu das placas de vídeo de "buffer de quadro" para as aceleradores gráficas, que trabalhavam diretamente com o Windows a fim de aumentar os tempos de resposta de tela e melhorar os gráficos em geral.

Neste período, as redes locais corporativas realmente começaram a decolar. A IBM promovia, então, o Office Vision, que deveria ser executado em todas as plataformas SAA, inclusive sobre o OS/2. E praticamente todos os gigantes do Desenvolvimento de sistemas tinham suas estratégias multiplataforma para a automação de escritórios, como o All-In-One da DEC. Quase todos fracassariam dentro de um espaço de tempo relativamente curto. Quem realmente alcançou o sucesso foram os servidores de PC, que abrigavam seus próprios dados e podiam fazer ligações com grandes bases de dados corporativas. No fronte do hardware, o Compaq Systempro, introduzido em 1989, liderava os grandes aplicativos que antes viviam em minicomputadores e outros grandes sistemas. No lado do software, chegava ao mercado o SQL, e companhias como a Oracle e a Sybase começavam a ter como alvos os desenvolvedores para PC. As ferramentas de *desenvolvimento rápido de aplicativos,* ou RAD, logo facilitaram a criação de boas interfaces com o usuário para o acesso a dados corporativos.

O correio eletrônico (email) é aceito no dia-a-dia das corporações com produtos, como o cc:Mail, mais tarde adquirido pela Lotus, e mais um punhado de concorrentes menores. Em dezembro de 1989, a Lotus mudou a fórmula com o Lotus Notes, o primeiro aplicativo de "groupware".

Em 1994, a Microsoft e a Intel já vestiam o manto da liderança na indústria do PC, o Windows tinha-se estabelecido como o padrão para aplicativos e as redes estavam definitivamente no mercado comum.

No começo de 1995, poderíamos esperar que novos sistemas operacionais da Microsoft e novos chips da Intel continuassem sendo o carro-chefe da computação ainda por muitos anos, levando-se em conta o histórico dos anos anteriores. Eles ainda são importantes, mas talvez a mudança mais importante destes últimos anos tenha vindo de um grupo de estudantes da Universidade de Illinois. Foi lá que, no início de 1993, Marc Andreessen, Eric

Bina e outros que trabalhavam para o National Center for Supercomputing Applications (NCSA) apareceram com o Mosaic, uma ferramenta que seria utilizada para paginar a Internet.

A Internet, é claro, já existia há muitos anos, datando do início dos anos 60, quando o órgão de Defesa de Projetos de Pesquisa Avançada (DARPA) do Pentágono estabeleceu as conexões com muitos computadores de universidades. Enquanto a Internet crescia, o governo transferiu seu controle para os sites individuais e comitês técnicos. E, em 1990, Tim Berners-Lee, então no laboratório de física CERN, em Genebra, Suíça, criou a Linguagem de Marcação de Hipertexto (HTML), uma maneira simples de ligar informações entre sites da Internet. Isto, por sua vez, gerou a World Wide Web (www), que apenas aguardava por um paginador gráfico para começar a crescer.

Após o lançamento do Mosaic ao público, no final de 1993, repentinamente, a Internet - e, em particular, a Web - podiam ser acessadas por qualquer pessoa que tivesse um computador pessoal, fato auxiliado, em parte, nela possibilidade de transferir livremente a versão mais recente de vários paginadores diferentes. E, dentro de pouco tempo, parecia que todo o mundo - e todas as companhias - estava inaugurando seu site na Web.

Novas versões de paginadores da Web também chegaram rapidamente. A Netscape Corp. - uma nova companhia formada por Andreessen e Jim Clark, que havia sido um dos fundadores da Silicon Graphics - logo começou a dominar o ramo de paginadores Web. O Netscape Navigator acrescentou vários recursos, inclusive o suporte a extensões (o que, por sua vez, levou a diversas extensões multimídia) e a máquina virtual Java (que permitia aos desenvolvedores elaborar aplicativos Java que podiam ser executados dentro do paginador).

A tremenda empolgação ocasionada pela explosão da World Wide Web chegou perto de eclipsar o maior anúncio da Microsoft neste período: o Windows 95. Introduzido em agosto de 1995, a estréia do software foi acompanhada por um entusiasmo maior do que qualquer outro anúncio de computação da era.

O Windows 95 era a versão do Windows pela qual muitos usuários estiveram esperando. Ele permitia a utilização de aplicativos totalmente 32 bits, tinha a multitarefa preemptiva, era compatível com Plug-and-Play, suportava novos padrões de e-mail e comunicações e, logicarnente, trazia uma nova interface com o usuário. Na verdade, muitos usuários pensavam que a nova interface, que incluía um menu "Iniciar" e uma área de trabalho de programas com pastas e ícones, deixaria o Windows muito mais próximo do projeto Lisa original ou do Macintosh de dez anos atrás.

A Microsoft passou anos prometendo um Windows 32 bits, chegando a dizer que ele estaria pronto em 1992, e os desenvolvedores passaram um longo tempo aguardando pelo "Chicago", como era conhecido o Windows 95 durante o desenvolvimento. Urna vez lançado, o Windows 95 rapidamente

tornou-se o padrão para a computação de usuário final, fazendo com que muitos desenvolvedores tivessem suas versões de aplicativos 32 bits prontas no lançamento do SO ou imediatamente após. A Microsoft fez seguir ao Windows 95, menos de um ano mais tarde, o Windows NT 4.0, que incorporava a mesma interface com o usuário e executava a maioria dos mesmos aplicativos, utilizando interfaces de programação Win32. O Windows NT agradou rapidamente os gerentes de IT corporativos, devido a seu projeto mais estável.

Mas ainda existe um grande espaço para avanços nos sistemas operacionais. Durante anos, os desenvolvedores de software falaram sobre as linguagens orientadas a objetos (como o C++) e sobre um sistema operacional mais orientado a objetos. Num projeto como este, dados e aplicativos deveriam ser divididos, para que os usuários pudessem trabalhar com os dados independentemente dos aplicativos individuais. O ideal seria que os dados pudessem ser disseminados ou distribuídos por diversos computadores.

A Microsoft vem falando sobre este conceito há anos, em especial na palestra "A Informação nas Pontas de Seus Dedos" de Bill Gates, realizada em novembro de 1990, que enfatizava o conceito de que todos os dados de que um usuário pode necessitar poderiam, algum dia, ser acessados por meio de um computador pessoal, independente do local onde os dados realmente residem. A idéia, disse ele, iria além dos aplicativos e consideraria apenas os dados. Este caminho levou à ênfase dada pela Microsoft aos documentos compostos, macros que funcionam através dos aplicativos, e a um novo sistema de arquivos. Algumas peças desta visão - chamada Cairo - fazem parte da interface do windows 95 e do OLE (Object Linking and Embedding). Outras ainda estão na prancheta de desenhos. É claro que os concorrentes da Microsoft continuaram seguindo seus próprios caminhos. Em 1989, a NEXT Computer de Steve Jobs apareceu com um SO orientado a objetos, destinado aos clientes corporativos e recentemente adquirido pela Apple Computer. No início dos anos 90, a IBM e a Apple fundiram dois de seus projetos - o SO "Pink" da Apple e o experimento IBM/Metaphor, chamado de Patriot Partners - para criar o Taligent. Este projeto resultou numa série um tanto extensa de estruturas, para uso dos desenvolvedores, na criação de aplicativos baseados em objetos. Mas, embora as estruturas tenham sido recentemente adicionadas ao OS/2, os planos para o Taligent como um SO isolado foram arquivados.

Uma outra tecnologia baseada em objetos está em vários estágios de desenvolvimento. O OLE, da Microsoft, que permite a criação de documentos compostos, tem sido aprimorado e hoje faz parte da especificação ActiveX, da mesma companhia. A Apple, a IBM e outras companhias surgiam com uma especificação alternativa chamada OpenDoc e tais componentes são hoje conhecidos como LiveObjects. A IBM definiu um padrão para que os objetos trabalhassem em conjunto ao longo de uma rede chamada Systems Object Model (SOM, ou Modelo de Objeto de Sistema), que concorre com o Component Object Model (COM, ou Modelo de Objeto Componente), da Microsoft.

Mas tudo isto tem sido eclipsado nos últimos meses pelo Java, da Sun Microsystems, que começou sua vida como uma variação do C++ projetada para uso na Internet. No ano passado, ele passou a incluir uma implementação de máquina virtual que foi incorporada aos paginadores da Netscape e da Microsoft e também à mais nova versão do sistema operacional da IBM, o OS/2 Warp. Muitos desenvolvedores estão atualmente desenvolvendo applets (pequenos aplicativos) e até mesmo aplicativos completos dentro do Java, na esperança de que isto venha a livrá-los de terem que se apoiar nos padrões Microsoft. Mais recentemente, a Sun, a Netscape e outras companhias estiveram promovendo a especifieação Java-Beans como um ótimo método de ligação de objetos.

Na própria Web, um dos atuais esforços são as tecnologias e produtos que permitem o fornecimento automático do conteúdo sobre a Internet, para que os usuários não precisem pesquisar informações específicas. Apresentada pela primeira vez pela PointCast, que implementou uma tela de descanso que coleta as informações da várias fontes, esta abordagem está sendo perseguida por diversos concorrentes novos, como a Castanet e a BackWeb. E tanto a Netscape quanto a Microsoft agora prometem o fornecimento de conteúdo Internet em segundo plano, com a Microsoft sempre falando em fazer disto uma parte da área de trabalho do Windows.

Quantas e quais destas iniciativas em equipamentos, programas e rede terão sucesso? Como sempre, é difícil dizer. Mas está claro que a Internet e a Web serão os principais fatores nos próximos anos, assim como os inevitáveis avanços nas capacidades de hardware e software.

1.3 Tipos de sistemas operacionais

Existem 4 tipos básicos de sistemas operacionais. Eles são divididos em grupos relacionados com o tipo de computador que controlam e o tipo de aplicativos que suportam. Estas são as categorias mais abrangentes:

- **sistema operacional de tempo real** (RTOS - Real-time operating system). É utilizado para controlar máquinas, instrumentos científicos e sistemas industriais. Geralmente um RTOS não tem uma interface para o usuário muito simples e não é destinado para o usuário final, desde que o sistema é entregue como uma "caixa selada". A função do RTOS é gerenciar os recursos do computador para que uma operação específica seja sempre executada durante um mesmo período de tempo. Numa máquina complexa, se uma parte se move mais rapidamente só porque existem recursos de sistema disponíveis, isto pode ser tão catastrófico quanto se uma parte não conseguisse se mover porque o sistema está ocupado.
- **monousuário, monotarefa**. O sistema operacional foi criado para que um único usuário possa fazer uma coisa por vez. O Palm OS dos computadores Palm é um bom exemplo de um moderno sistema operacional monousuário e monotarefa.
- **monousuário, multitarefa**. Este tipo de sistema operacional é o mais utilizado em computadores de mesa e laptops. As plataformas

Microsoft Windows e Apple MacOS são exemplos de sistemas operacionais que permitem que um único usuário utilize diversos programas ao mesmo tempo. Por exemplo, é perfeitamente possível para um usuário de Windows escrever uma nota em um processador de texto ao mesmo tempo em que faz download de um arquivo da Internet e imprime um e-mail.

- **multiusuário**. Um sistema operacional multiusuário permite que diversos usuários utilizem simultaneamente os recursos do computador. O sistema operacional deve se certificar de que as solicitações de vários usuários estejam balanceadas. Cada um dos programas utilizados deve dispor de recursos suficientes e separados, de forma que o problema de um usuário não afete toda a comunidade de usuários. Unix, VMS e sistemas operacionais mainframe como o MVS são exemplos de sistemas operacionais multiusuário.

Tela do sistema operacional Mac OS X Panther

É importante diferenciar os sistemas operacionais multiusuário dos sistemas operacionais monousuário que suportam rede. O *Windows Server e o Novell Open Enterprise Server* podem suportar centenas ou milhares de usuários em rede, mas os sistemas operacionais em si não são sistemas multiusuário de verdade. O **administrador do sistema** é o único "usuário" do *Windows Server* ou do *Novell Open Enterprise Server*. O suporte à rede e todos os usuários remotos são, do ponto de vista do sistema operacional, um programa sendo executado pelo administrador.

Agora que você conhece os tipos de sistemas operacionais, vamos entender as suas funções básicas.

2 – Conceitos sobre sistemas operacionais

2.1 – Processos e threads

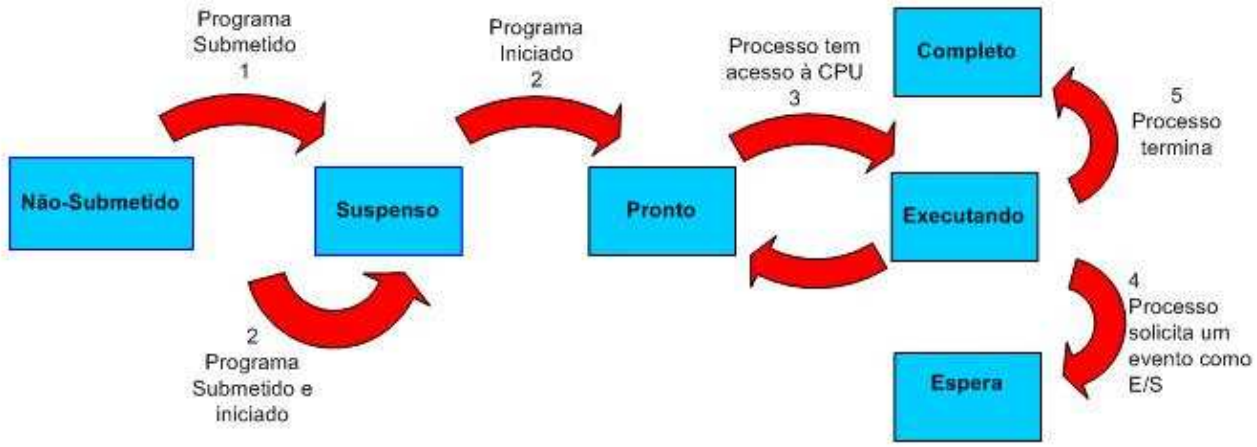
Processos

Processo, no contexto da informática, é um programa de computador em execução.

Em sistemas operacionais, processo é um módulo executável único, que corre concorrentemente com outros módulos executáveis. Por exemplo, em um ambiente multi-tarefa (como o Unix) que suporta processos, um processador de texto, um navegador e um sistema de banco de dados são processos separados que podem rodar concomitantemente. Processos são módulos separados e carregáveis, ao contrário de threads, que não podem ser carregadas. Múltiplas threads de execução podem ocorrer dentro de um mesmo processo. Além das threads, o processo também inclui certos recursos, como arquivos e alocações dinâmicas de memória e espaços de endereçamento.

A comunicação entre processos é o grupo de mecanismos que permite aos processos transferirem informação entre si. A capacidade de um sistema operacional executar simultaneamente dois ou mais processos é chamada multiprocessamento. Se existirem dois ou mais processos executados em simultâneo e disputam o acesso a recursos partilhados, problemas da concorrência podem ocorrer. Estes problemas podem ser resolvidos pelo gerenciamento adequado de múltiplas linhas de execução ou processos através da sincronização (multitarefa) ou por outros recursos (como a troca de contexto).

Estados de processos



Estado 1 - Não-Submetido

É o processo que ainda não está submetido a CPU, está nas mãos do usuário." Até onde interessa ao sistemas ele não existe, porque o usuário ainda não o submeteu. Ele é simplesmente apresentado como sendo o primeiro passo na vida de um processo. O Sistema Operacional, naturalmente, não reconhece esse estado.[1]". Pode por exemplo, ser um arquivo executável que está armazenado no HD.

Estado 2 - Suspenso

É o processo que já foi submetido, porém permanece suspenso até que o horário ou evento programado ao usuário venha acontecer. Temos como exemplo o agendamento de uma varredura programada no anti-vírus por um usuário.

Estado 3 - Pronto

O processo já foi submetido e está pronto para receber a CPU, porém ainda guarda o escalonador de processos para ter controle da CPU. Processos que estão esperando E/S não se aplicam a esse estado.

Estado 4 - Executando

A execução propriamente dita. O código está sendo processado. Se ocorrer durante a execução uma requisição de E/S o processo é colocado no estado de espera e outro processo da fila de prontos poderá então concorrer a CPU.

Estado 5 - Espera

É o processo que foi colocado na fila de espera de E/S devido ao processador de E/S ser mais lento que a CPU principal. O processo tornaria a CPU mais escrava dele se não houvesse esse estado, pois como ele não está concorrendo à CPU ao executar um E/S, pode-se então colocá-lo no estado de espera para que os demais processos do estado pronto possam concorrer a CPU.

Ex: parte de um código em C
```
scanf(“%d”, VALOR);
SOMA=VALOR+JUROS;
```

Como podemos notar, a instrução scanf (uma requisição de entrada e saída) é gerada se não fosse possível colocar o processo em estado de espera; caso o usuário não entrasse com nenhum valor, o programa ficaria suspenso e não liberaria a CPU para outros processos.

Estado 6 - Completo

Neste estado temos a finalização do processo.

Threads

Da mesma forma que os processos sofrem escalonamento as threads também tem a mesma necessidade. O escalonamento de threads é variável dependedo do tipo da thread que são Kernel-Level Thread e User-Level Thread. Da mesma forma que quando vários processos são executados em apenas uma CPU eles sofrem escalonamento e parecem que todos são executados ao mesmo tempo, quando um processo tem threads elas esperam a sua vez para ser executadas, como esta alternância é muito rápida há impressão de que todos os processos e as thread destes processos são executadas paralelamente.

Geralmente quando iniciamos um processo com múltiplas threads existe um thread principal que é responsável por gerenciar criar novas threads , quando uma thread finaliza seu trabalho ela entra em thread_yield, que sinaliza que a thread encerrou seu trabalho e esta liberando a CPU para outra thread ou processo.

User-Level Thread

As ULT(User-Level Thread) são escalonadas pelo programador, tendo a grande vantagem de cada processo ter como usar um algoritmo de escalonamento que melhor se adapte a situação, o SO não tem a obrigação de fazer o escalonamento destas threads, em geral ele nem sabe que as threads existem, estas threads são geralmente mais rápidas que as KLT, pois dispensam a chamada ao SO para escalonar evitando assim uma mudança total de contexto do processador memória e diversas outros níveis para alternar os processos. Neste modo o programador é responsável por criar, executar, escalonar e destruir a thread. Vamos a um exemplo prático, um processo chamado P1, este processo pode ter varias threads, neste exemplo ele tem P1T1, P1T2, P1T3, então quando o SO da o CPU para o processo P1 cabe a ele destinar qual thread será executada, caso esta thread use todo processo do quantum, o SO chamara outro processo, e quando o processo P1 voltar a executar P1T1 voltará a ser executada e continuará executando até seu término ou intervenção de P1, este comportamento não afetará outros processos pois o SO continua escalonando os processos normalmente.

Kernel-Level Thread

As KLT são escalonadas diretamente pelo SO, comumente são mais lentas que as Threads ULT pois a cada chamada elas necessitam consultar o SO, exigindo assim a mudança total do contexto do processador memória e outros níveis necessários para alternar um processo. Vamos citar outro exemplo pratico, um processo chamado P2 com as threads P2T1, P2T2, P2T3, e outro processo chamado P3 com as threads P3T1, P3T2, P3T3. SO não entregara a CPU ao processo e sim a uma thread deste processo, note agora que o SO é

responsável por escalonar as threads e este SO tem que suportar Threads, a cada interupção de thread é necessário mudar todo o contexto de CPU e memória porém as threads são independentes dos processos, podemos executar então P3T2, P2T1, P2T2, P2T1, P3T1,P2T3,P3T3, ou seja a ordem em que o escalonador do SO determinar. Já com as threads em modo usuário não conseguimos ter a mesma independência, pois quando passamos o controle ao processo enquanto seu quantum for válido ele irá decidir que thread irá rodar. Um escalonamento típico do SO é onde o escalonador sempre escolhe a thread de maior prioridade, que são divididas em duas classes que são Real Time e Normal, cada thread ganha uma prioridade ao ser criada que varia de 0 a 31(0 é a menor e 31 maior), processos com prioridade 0 a 15(Real Time) tem prioridade ajustada no tempo de execução como nos processos de E/S que tem a prioridade aumentada variando o periférico, processos com prioridade 16 a 31 são executados até terminar e não tem prioridade alterada, mas somente uma thread recebe a prioridade zero que é a responsável por zerar páginas livres no sistema. Existe ainda uma outra classe chamada de idle, uma classe mais baixa ainda, só é executada quando não existem threads aptas, threads dessa classe não interferem na performance.

2.2 – Deadlocks

Deadlock (blocagem, impasse), no contexto do sistemas operacionais (SO), caracteriza uma situação em que ocorre um impasse e dois ou mais processos ficam impedidos de continuar suas execuções, ou seja, ficam bloqueados. Trata-se de um problema bastante estudado no contexto dos Sistemas Operacionais, assim como em outras disciplinas, como banco de dados, pois é inerente à própria natureza desses sistemas.

O deadlock ocorre com um conjunto de processos e recursos não-preemptíveis, onde um ou mais processos desse conjunto está aguardando a liberação de um recurso por um outro processo que, por sua vez, aguarda a liberação de outro recurso alocado ou dependente do primeiro processo.

A definição textual de deadlock normalmente, por ser muito abstrata, é mais difícil de se compreender do que a representação por grafos, que será resumida mais adiante. No entanto, algumas observações são pertinentes:

- O deadlock pode ocorrer mesmo que haja somente um processo no SO, considerando que este processo utilize múltiplos threads e que tais threads requisitem os recursos alocados a outros threads no mesmo processo;

- O deadlock independe da quantidade de recursos disponíveis no sistema;

- Normalmente o deadlock ocorre com recursos como dispositivos, arquivos, memória etc. Apesar da CPU também ser um recurso para o SO, em geral é um recurso facilmente preemptível, pois existem os

escalonadores para compartilhar o processador entre os diversos processos, quando trata-se de um ambiente multitarefa.

Um exemplo onde erros de deadlock ocorrem é no banco de dados. Suponha que uma empresa tenha vários vendedores e vários pontos de venda/caixas. O vendedor A vendeu 1 martelo e 1 furadeira. O sistema então solicita o travamento do registro da tabela ESTOQUE que contém o total de martelos em estoque e em seguida solicita o travamento do registro que contém o total de furadeiras em estoque. De posse da exclusividade de acesso aos dois registros, ele lê a quantidade de martelos, subtrai 1 e escreve de novo no registro, o mesmo com o registro de furadeiras. Observe, no entanto que existem diversos caixas operando simultaneamente de forma que se algum outro caixa naquele exato instante estiver vendendo um furadeira, ele ficará de aguardando a liberação do registro das furadeiras para depois alterá-lo. Note que ele só altera os registro depois que for dada exclusividade para ele de TODOS os recursos que ele precisa, ou seja, de todos os registro. Suponha agora que em outro caixa a venda foram vendidos 1 furadeira e 1 martelo e que o outro caixa solicitou o travamento do registro com a quantidade de furadeiras e agora quer o acesso ao de martelos, no entanto o de martelos está travado para o primeiro caixa. Nenhum deles devolve o recurso (registro) sobre o qual tem exclusividade e também não consegue acesso ao outro registro que falta para terminar a operação. Isto é um deadlock.

Condições necessárias para a ocorrência de deadlock

No texto acima, foi dito que o deadlock ocorre naturalmente em alguns sistemas. No entanto, é necessário ressaltar que tais sistemas precisam obedecer a algumas condições para que uma situação de deadlock se manifeste.

Essas condições estão listadas abaixo, onde as três primeiras caracterizam um modelo de sistema, e a última é o deadlock propriamente dito: processos que estejam de posse de recursos obtidos anteriormente podem solicitar novos recursos. Caso estes recursos já estejam alocados a outros processos, o processo solicitante deve aguardar pela liberação do mesmo;

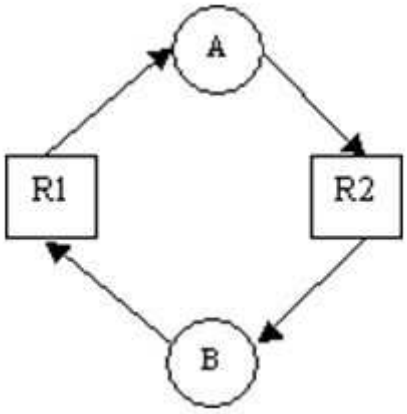
- Condição de não-preempção: recursos já alocados a processos não podem ser tomados a força. Eles precisam ser liberados explicitamente pelo processo que detém a sua posse;

- Condição de espera circular: deve existir uma cadeia circular de dois ou mais processos, cada um dos quais esperando por um recurso que está com o próximo membro da cadeia.

- Condição de exclusão mútua: cada recurso ou está alocado a exatamente um processo ou está disponível;

- Condição de posse-e-espera: cada processo pode solicitar um recurso, ter esse recurso alocado para si e ficar bloqueado esperando por um outro recurso;

Representação de deadlock em grafos



Exemplo de representação de deadlock em grafos de alocação de recursos, com dois processos A e B, e dois recursos R1 e R2.

O deadlock também pode ser representado na forma de grafos dirigidos, onde o processo é representado por um círculo e o recurso, por um quadrado. Quando um processo solicita um recurso, uma seta é dirigida do círculo ao quadrado. Quando um recurso é alocado a um processo, uma seta é dirigida do quadrado ao círculo.

Na figura do exemplo, podem-se ver dois processos diferentes (A e B), cada um com um recurso diferente alocado (R1 e R2). Nesse exemplo clássico de deadlock, é facilmente visível a condição de espera circular em que os processos se encontram, onde cada um solicita o recurso que está alocado ao outro processo.

Tratamento de deadlock

As situações de deadlock podem ser tratadas ou não em um sistema, e cabe aos desenvolvedores avaliar o custo/benefício que essas implementações podem trazer. Normalmente, as estratégias usadas para detectar e tratar as situações de deadlocks geram grande sobrecarga, podendo até causar um dano maior que a própria ocorrência do deadlock, sendo, às vezes, melhor ignorar a situação.

Existem três estratégias para tratamento de deadlocks:

- **Ignorar a situação;**
- **Detectar o deadlock e recuperar o sistema; e**
- **Evitar o deadlock;**

Algoritmo do Avestruz (Ignorar a situação)

A estratégia mais simples para tratamento (ou não) do deadlock, conhecida como Algoritmo do Avestruz, é simplesmente ignorá-lo. Muitos defendem que a

freqüência de ocorrência deste tipo de evento é baixa demais para que seja necessário sobrecarregar a CPU com códigos extras de tratamento, e que, ocasionalmente, é tolerável reiniciar o sistema como uma ação corretiva.

Detectar o deadlock e recuperar o sistema

Nessa estratégia, o sistema permite que ocorra o deadlock e só então executa o procedimento de recuperação, que resume-se na detecção da ocorrência e na recuperação posterior do sistema. É na execução desse procedimento que ocorre a sobrecarga, pois existem dois grandes problemas: primeiramente, como/quando detectar o deadlock e depois, como corrigi-lo.

Para detectar o deadlock, o sistema deve implementar uma estrutura de dados que armazene as informações sobre os processos e os recursos alocados a eles. Essas estruturas deverão ser atualizadas dinamicamente, de modo que reflitam realmente a situação de cada processo/recurso no sistema.

Só o mero procedimento de atualização dessas estruturas já gera uma sobrecarga no sistema, pois toda vez que um processo aloca, libera ou requisita um recurso, as estruturas precisam ser atualizadas.

Além disso, o SO precisa verificar a ocorrência da condição de espera circular nessas estruturas para a efetiva detecção do deadlock. Esse procedimento, por sua vez, gera outra sobrecarga, que pode ser mais intensa se não for definido um evento em particular para ser executado, como a liberação de um recurso, por exemplo. Assim, ou o SO verifica periodicamente as estruturas (o que não é aconselhável, pois pode aumentar consideravelmente o tempo de espera dos processos não-bloqueados), ou pode-se implementar uma política, onde o SO verifica as estruturas quando o mesmo realizar algum procedimento de manutenção do sistema, por exemplo.

Finalmente, só após detectar a presença do deadlock no sistema, o SO precisa corrigi-lo, executando um procedimento de recuperação.

Quanto à detecção do deadlock, vamos apresentar uma das técnicas usadas para detectar a ocorrência de deadlock em sistemas que possuem vários recursos de cada tipo.

Detecção de deadlock com vários recursos de cada tipo

O algoritmo de detecção de deadlock com vários recursos de cada tipo baseia-se em um ambiente que possua vários recursos do mesmo tipo e os processos solicitam apenas pelo tipo de recursos, não especificando qual recurso desejam utilizar.

Assim, um processo pode requisitar uma unidade de CD para leitura. Se o sistema possuir duas, o processo pode utilizar a que estiver disponível, em vez de especificar uma delas. Dessa forma, o processo solicita o recurso pelo tipo, sem discriminação.

O algoritmo para essa detecção trabalha com duas variáveis, três matrizes unidimensionais (vetores) e duas matrizes bidimensionais, descritas a seguir:

- **Estruturas:**
  - n: Variável inteira. Representa a Quantidade de Processos Ativos;
  - m: Variável inteira. Representa a Quantidade de Tipos de Recursos;
  - Matriz E = $(e_j)_m$: Matriz unidimensional, de tamanho m. Representa a Matriz de Recursos Existentes;
  - Matriz A = $(a_j)_m$: Matriz unidimensional, de tamanho m. Representa a Matriz de Recursos Atualmente Disponíveis;
  - Matriz W = $(w_j)_m$: Matriz unidimensional, de tamanho m. Representa uma Matriz Auxiliar, presente somente para facilitar o cálculo durante a execução do algoritmo;
  - Matriz C = $(c_{ij})_{nxm}$: Matriz bidimensional, de tamanho n x m. Representa a Matriz de Alocação Corrente;
  - Matriz R = $(r_{ij})_{nxm}$: Matriz bidimensional, de tamanho n x m. Representa a Matriz de Recursos Requisitados.

- **Faça (para preenchimento das estruturas):**
  1. Preencher a Matriz E com as quantidade de instâncias de cada tipo de recurso;
  2. Preencher a Matriz C com as quantidade de instâncias de cada tipo alocadas aos processos, sendo que a somatória de cada coluna da Matriz C deve ser menor ou igual à quantidade do recurso correspondente na Matriz E (os processos nunca podem requisitar mais recursos que existentes no sistema);
  3. Preencher a Matriz W com o resultado da subtração da quantidade de cada recurso da Matriz E com o valor do somatório de cada coluna do recurso correspondente da Matriz C, ou seja:

$$w_j = e_j - \sum_{i=1}^{n} c_{ij}$$

  4. Preencher inicialmente a Matriz A com os valores da Matriz W. Note que: $a_j = e_j + w_j$;
  5. Preencher a Matriz R com as próximas requisições dos processos, seguindo as mesmas regras da Matriz C.

- **Faça (para detecção do deadlock):**
  1. Inicialmente, desmarcar todos os processos;

2. Para um processo Pi desmarcado, verificar se todos os elementos da linha i na Matriz R são menores ou iguais aos da Matriz A;
3. Se for, então marque o execute o processo Pi e libere os recursos requisitados pelo processo na Matriz C (adicionar a linha i da Matriz C na Matriz A);
4. Retornar ao passo 2.

A lógica do algoritmo é a seguinte: cada processo é considerado como apto a ser executado até que a detecção prove o contrário. A detecção apenas verifica se os processos requisitam mais recursos do que estão disponíveis, o que caracteriza um deadlock. Caso o processo requisite uma quantidade disponível, então ele pode ser executado, e os recursos que foram solicitados antes podem também ser liberados de volta ao sistema, o que pode permitir que outros processos também concluam suas execuções e liberem os recursos.

Um exemplo do preenchimento das matrizes encontra-se na figura abaixo, considerando-se n=2 e m=3.

Matriz E			
Recurso	**1**	**2**	**3**
Quantidade	2	4	3

Matriz A			
Recurso	**1**	**2**	**3**
Quantidade	1	3	2

Matriz C			
Recurso	**1**	**2**	**3**
Processo 1	1	1	1
Processo 2	0	0	0

Status
Deadlock
Ok

Matriz R			
Recurso	**1**	**2**	**3**
Processo 1	2	4	3
Processo 2	1	2	1

Exemplo do preenchimento das matrizes do algoritmo de detecção de deadlock com vários recursos de cada tipo.

2.3 - **Gerenciamento de memória**

Gerenciamento de memória é um complexo campo da ciência da computação e são constantemente desenvolvidas várias técnicas para torná-la mais eficiente. Em sua forma mais simples, está relacionado em duas tarefas essenciais:

- **Alocação**: Quando o programa requisita um bloco de memória, o gerenciador o disponibiliza para a alocação;
- **Reciclagem**: Quando um bloco de memória foi alocado, mas os dados não foram requisitados por um determinado numero de ciclos, esse é liberado e pode ser reutilizado para outra requisição.

Gerência de Memória

A cada dia que passa os programadores necessitam de mais memória e mais programas rodando simultaneamente para poderem tratar cada vez mais informações. O tratamento necessário da memória utilizada não é uma tarefa fácil de ser implementada. Existem vários requisitos que devem ser observados para o correto funcionamento, tais como, Segurança, Isolamento, Performance, entre outros. Para isto a função de gerenciar a memória passa a ser do sistema operacional e não mais do aplicativo.

Alocação

A alocação de memória pode ser:

- Alocação estática: Decisão tomada quando o programa é compilado.
- Alocação dinâmica: Decisão é adiada até a execução. (Permite swapping)

Fragmentação

Desperdício de páginas de memória alocadas

Pode ser de dois tipos: interna e externa. Interna: Ocorre quando o processo não ocupa inteiramente os blocos de memória (páginas) reservados para ele. Geralmente acontece pois o tamanho do processo não é um múltiplo do tamanho da página de memória, o que acarreta sobra de espaço na última página alocada.

Externa: Ocorre à medida que os programas vão terminando e deixando lacunas cada vez menores de espaços, o que os torna inutilizáveis.

Estratégias para “atacar” o problema com o algoritmos First-fit, Best-fit, Worst-fit e Next-fit

Paginação

“Quebra” a memória do processo permitindo espaços de endereçamento não contíguos.

TLB

A Translation Lookaside Buffer (TLB) é um conjunto de registradores especiais que são super rápidos. Cada registrador tem duas partes: chave e valor. Dada uma chave, busca-se o valor correspondente. Geralmente, 64 entradas, no máximo, e a busca é feita em todos os registradores simultaneamente.

Memória Virtual é uma ténica poderosa e sofisticada de gerência de memória, onde as memórias principal e secudária são combinadas, dando ao usuário a ilusão de existir uma memória muito maior que a capacidade real da memória principal. O conceito desta técnica fundamenta-se em não vincular o

endereçamento feito pelo programa aos endereços físicos da memória principal. Desta forma, programas e suas estruturas de dados deixam de estar limitados ao tamanho da memória física disponível, pois podem possuir endereços associados à memória secundária.

Algoritmos de Substituição de Página

- Algoritmo Ótimo
- Algoritmo Não Usada Recentemente
- Algoritmo FIFO
- Algoritmo Segunda Chance
- Algoritmo do relógio
- Menos Recentemente Usada
- WSClock

Gerenciamento manual de memória

Em modelos de gerenciamento manual, podem ocorrer os problemas conhecidos como *vazamento de memória*, que acontece quando uma quantidade de memória é alocada e não é liberada ainda que nunca seja utilizada. Isto ocorre quando objetos perdem a referência sem terem sido liberados, mantendo o uso do espaço de memória.

Garbage Collector

É o gerenciamento automático de memória, também conhecido como *coletores*, sendo conhecido em Portugal como *reciclagem automática de memória*. Este serviço libera os blocos de memória que não sejam mais usados por um programa automaticamente.

As vantagens desse tipo de gerenciamento são:

- Liberdade do programador: Não é obrigado ficar atento aos detalhes da memória;
- Menos bugs de gerenciamento de memória: Por se tratar de uma técnica mais confiável;
- Gerenciamento automático: Mais eficiente que o manual;

E entre as desvantagens, podemos citar:

- O desenvolvedor tende a estar mais desatento em relação a detalhes de memória;
- O gerenciador automático ainda apresenta limitações.

Quando deixam de existir referências a um objeto, este passa a ser considerado apto a ser "coletado" pelo garbage collector, que significa dizer que será removido da memória, deixando-a livre para uso por outros objetos.

Os algoritmos de *garbage collection* operam de um modo que permite classificá-los em duas grandes famílias:

- identificação directa: por contagem de referências (*reference counting*);
- identificação indirecta: por varrimento (*tracing*), que pode incluir também compactação da memória livre; cópia; ou geracional (utilizado nas máquinas virtuais Java e .Net)

Gerenciamento de memória no DOS

O IBM PC original foi projetado com uma memória RAM de 1024KB

- 640KB
  - Para o sistema operacional e programas
- 384KB - *área de memória superior* (Upper Memory Area) ou UMA
  - Para os adaptadores diversos como EGA & VGA, MDA, adaptadores CGA, e de redes.
  - ROM BIOS e Shadow RAM.
  - E mais tarde, área de paginação de expansão de memória (EMS) vista mais adiante.

Logo depois, foi provado que esta quantidade de memória se tornaria insuficiente para as necessidades futuras.

Entretanto, os sistemas operacionais e aplicativos desenvolvidos até então não seriam capazes de reconhecer um endereço de memória superior aos 640KB originais, o que levou os projetistas a desenvolverem ferramentas que executariam esta tarefa.

Emm386.exe

É o dispositivo de instalação da *memória expandida* (Expanded Memory) ou EMS. A EMS consistia em toda a memória acima dos 1024KB (1MB) original, em computadores baseados nas tecnologias dos processadores 80286, 80386, i486 ou Pentium. A capacidade máxima de endereçamento fica limitada a 32MB e seu acesso é através de uma paginação de 64KB na área UMA. Os programas deveriam ser escritos de forma a poderem reconhecer a área EMS.

O nome "EMM" vem do inglês *Extended Memory Manager*.

Himem.sys

É o dispositivo de instalação da *área de memória alta* (High Memory Area), conhecida também como HMA. Sua principal função é controlar o uso da memória estendida do computador, de modo que dois ou mais aplicativos ou dispositivos não utilizem o mesmo endereço de memória ao mesmo tempo. Para as primeiras versões do Windows, o Himem.sys fornecia suporte para que este fosse executado em modo protegido.

O nome Himem vem do inglês *High Memory*.

Smartdrv.exe

É o gerenciador de memória cache de disco (no Novell DOS 7, é chamado de Nwcache.exe). Ambos possuem a mesma função que é a de minimizar o acesso ao disco, carregando um bloco de informações na memória para processamento, ao invés de ir buscar aos poucos no disco. Existiram também placas de expansão chamadas de *Disk Accelerator*, que possuem um hardware próprio para esta tarefa, como por exemplo o 'Disk Accelerator PROMISE DC4030VL. Atualmente, esta técnica é desempenhada por memórias localizadas na placa principal do sistema (*cache on-board*) resultando, portanto, dois modos de gerenciamento de memória cache de disco: por *software* e por *hardware*.

O nome Smartdrv é uma abreviação do inglês *Smart Drive*.

2.5 – Entrada e Saída

GERÊNCIA DE ENTRADA E SAÍDA
Uma das principais funções de um SO é controlar os dispositivos de entrada e saída(E/S), fornecendo uma interface de uso adequada. Esse capítulo mostra como o SO interage com os dispositivos de E/S, enviando comandos e capturando suas interrupções. A gerência de E/S está intimamente relacionada com os aspectos de hardware de um computador.

Princípios básicos de hardware

Um **periférico** (ou **dispositivo de E/S**) é qualquer dispositivo conectado a um computador de forma a possibilitar a interação do computador com o mundo externo. Atualmente, existe uma grande variedade de dispositivos, desde aqueles desenvolvidos
para permitir a comunicação do homem com o computador (teclado, mouse, monitor de vídeo, etc) até dispositivos que possibilitam a comunicação entre computadores (modems, placas de redes, etc), ou ainda aqueles destinados ao armazenamento de informações (unidades de fita, disquetes, disco rígido, CD-ROM, etc). Apesar dessa diversidade, existe uma razoável padronização na forma de acessar (acionar) esses periféricos.
De acordo com o sentido do fluxo de dados entre o computador e o dispositivo, esses podem ser divididos em *periféricos de entrada*, *periféricos de saída*, ou ainda *periféricos de entrada e saída*.
Um periférico é conectado ao computador através de um componente de hardware denominado **interface**. Essa, por sua vez, é interconectada aos barramentos internos do computador. Para tratar a diversidade, a complexidade, e as diferentes formas de operações de cada periférico, as interfaces empregam no seu projeto um outro componente, o **controlador**, que nada mais é que um processador projetado especificamente para realizar uma função, como, por exemplo, controlar um disco rígido.
A UCP se comunica com o controlador através de um conjunto de registradores situados na interface. Tipicamente, são 3 registradores: **registrador de dado**, **registrador de status** e **registrador de comando**. Por exemplo, para escrever um dado em um dispositivo de saída, seriam realizadas as seguintes duas operações: (1) UCP coloca o dado no registrador de dados, (2) UCP coloca comando *write* no registrador de comando. A partir daí, o controlador comanda

as ações do periférico, de forma independente da UCP. No final das ações, o controlador coloca o resultado da operação (sucesso ou falha) no registrador de status e gera uma interrupção. O caso de não haver mecanismo de interrupção é considerado adiante.
A função básica de um controlador é implementar um conjunto de comandos para o seu dispositivo. O controlador vai traduzir cada ordem colocada no registrador de comando numa seqüência específica de acionamentos eletrônicos, elétricos e mecânicos que irão realizar a operação solicitada. Evidentemente, cada tipo de periférico necessita de um controlador diferente. Contudo, apesar da grande variedade de dispositivos, não dá para negar que existe uma boa padronização na forma de conectá-los ao computador e na forma de acioná-los através da UCP.
Os dispositivos de E/S, dependendo de sua interconexão fisica às interfaces, podem ser do tipo **serial** ou **paralelo**. Uma **interface serial** é aquela em que existeapenas uma linha para transmitir dados. Nesse caso, um byte vai ser transferido como uma seqüência de bits. Os modems, alguns tipos de *mouses* e impressoras são exemplos de dispositivos seriais. Uma **interface paralela** possui várias linhas para os dados, permitindo assim que vários bits sejam transferidos simultaneamente (em paralelo) entre o dispositivo de E/S e a interface. O número de linhas corresponde normalmente ao número de bits que compõem um byte (*8 bits*) ou palavra (*n x 8 bits*). As impressoras paralelas são exemplos típicos desse tipo de dispositivo.

• Acesso aos registradores dos periféricos

Conforme visto anteriormente, a UCP “enxerga” um periférico através de um conjunto de registradores, os quais registram ordens, fornecem o estado de uma operação, e permitem a leitura (escrita) de dados do (no) periférico. Esses registradores residem fisicamente no hardware da interface e podem ser acessados pela UCP através de duas técnicas distintas: **entrada e saída mapeada em espaço de E/S** ou **entrada e saída mapeada em memória**. A opção por uma das técnica é feita em tempo de projeto do computador. Na primeira (**mapeamento em espaço de E/S**), o processador possui instruções de máquina próprias para operações de E/S e estas instruções especificam registradores de periféricos. Nesse tipo de processador existem dois espaços de endereçamento distintos: o espaço normal, que corresponde à memória principal e o espaço de E/S, que corresponde aos registradores dos periféricos. Um conjunto de instruções (do estilo *load*, *store*, *move*) acessa a “memória normal” e outro conjunto (as instruções de E/S, do estilo *in* e *out*) acessa a “memória de E/S”. Enquanto a instrução *move end,dado* representa escrever *dado* na posição de memória *end*, a instrução *out end,dado* representa escrever *dado* no registrador de algum dispositivo. Observe que, numericamente, o valor de *end* pode ser o mesmo nos dois casos, o que faz a diferença é a instrução empregada.

No **mapeamento em memória**, o espaço de endereçamento é único (só existe uma memória), mas alguns endereços representam registradores de (controladores de) periféricos. Em tempo de projeto do computador, são reservados alguns endereços da memória principal para representar registradores de periféricos. Esses endereços não corresponderão a palavras

de memória, mas sim a registradores de dispositivos. Com essa técnica, qualquer periférico pode ser programado através de instruções de acesso à memória do estilo *mov end,dado*. Se *end* é um endereço correspondente a um controlador de dispositivo, o dado será escrito no registrador do dispositivo (o valor de *dado* será interpretado segundo a função específica desse registrador).

• Interação entre a UCP e os controladores de periféricos

Independentemente do tipo de mapeamento utilizado, a interação entre a UCP e a interface (controlador), para realizar operações de E/S, pode acontecer de três maneiras diferentes: “E/S programada”, “via interrupções” e “acesso direto à memória (DMA)”.

E/S programada

Essa técnica é usada quando não há sistema de interrupção (nos computadores antigos era assim, hoje a técnica só é usada em máquinas simples). Com esta técnica, toda interação entre a UCP e o controlador é de responsabilidade do programador. O ciclo de funcionamento é baseado no envio de um comando ao controlador e na espera de sua realização. Por exemplo, o processador envia um comando de leitura ao controlador e, em seguida, fica testando continuamente (em *busy loop*) o registrador de estado para verificar se o dado solicitado já está disponível. Em caso afirmativo, o processador efetua a leitura.
O problema desse método é que as operações de E/S são muito lentas em comparação com as operações de cálculo. Utilizar continuamente o processador para verificar o andamento de uma operação de E/S representa um desperdício muito grande de tempo de cálculo. Uma solução paliativa é inserir operações de cálculo entre as verificações sucessivas do estado da operação de E/S. Esse procedimento de verificar periodicamente o estado de uma operação de E/S é denominado de ***polling***. O emprego de *polling* reduz o desperdício de tempo do processador, mas introduz outro problema: determinar a freqüência de sua realização. Se a freqüência é muito alta, há desperdício de tempo, principalmente se o dispositivo é realmente lento. Por outro lado, efetuar o polling com uma freqüência baixa pode acarretar esperas desnecessárias por parte do dispositivo de E/S, ou ainda, o que é pior, perda de informações (este problema existe nas interfaces de comunicação que possuem *buffers* de tamanho limitado para a recepção de dados: a não leitura dos dados recebidos em um certo período de tempo pode acarretar perdas por “sobre-escrita”). O ideal seria que o dispositivo de E/S sinalizasse o processador logo que o dado estivesse disponível.

Comunicação via interrupção

O uso do mecanismo de interrupção elimina a necessidade de *polling* em operações de E/S. Nesse caso tem-se E/S via interrupções (*interrupt driven I/O*). Nessa situação, o processador é responsável – via software – apenas por iniciar uma operação de E/S enviando comandos à interface (controlador). Após, o processador passa a executar outra tarefa, e o controlador, a operação

de E/S. Quando a operação de E/S termina, o controlador interrompe o processador, provocando a execução do tratador de interrupção, o qual irá acionar o driver do dispositivo.

Acesso Direto à Memória

O emprego de interrupções resolve o problema de determinar o momento exato em que um dispositivo de E/S necessita da atenção do processador, entretanto não auxilia na execução de outra tarefa bastante importante em operações de E/S: a transferência de dados.
Para a maioria dos periféricos, o processador é responsável pela leitura de dados na interface e por sua transferência à memória, ou pela leitura na memória e transferência para a interface. Quando o volume de dados é importante, esse procedimento torna-se ineficiente, pois, além de envolver o processador, essa transferência representa dois movimentos de dados: um da interface ao processador, e outro do processador à memória. Uma solução para isso é transferir diretamente os dados da interface para a memória, o que é conseguido por um mecanismo conhecido como **DMA** (*Direct Memory Access*).
A técnica de DMA baseia-se no emprego de um hardware especial, o controlador de DMA, para realizar a transferência de dados entre um dispositivo e a memória. Para tanto, o controlador de DMA possui a capacidade de acessar diretamente a memória, sendo então conectado fisicamente ao barramento de dados e de endereços do computador. O controlador de DMA possui internamente uma série de registradores utilizados pela UCP para programar a transferência de dados. Tipicamente, tem um par de registradores para o armazenamento dos endereços fonte e destino da transferência, um registrador que determina quantos bytes devem ser transferidos, um registrador de comando e um de estado. Após acionar o DMA, o processador pode se dedicar a outra tarefa. No término da transferência, o controlador de DMA sinaliza o processador através de uma interrupção de hardware.
A técnica de DMA é mais eficiente que as discutidas anteriormente quando a operação de E/S envolve a leitura (ou escrita) de muitos dados, como, por exemplo, uma leitura de disco ou a recepção de uma mensagem em uma rede local. É importante observar que tanto o controlador de DMA quanto o processador competem para acessar à memória (não esqueça que o processador está executando outras tarefas, o que envolve sem dúvida acessos à memória). Essa disputa pelo acesso à memória é coordenada pelo que se denomina **arbitramento do barramento**. Como o mecanismo de acesso à memória passa a ser compartilhado, o processador (durante o funcionamento do DMA) irá executar a uma velocidade menor que a normal. Mesmo assim, esse método será ainda mais eficiente do que usar diretamente o processador para realizar a transferência via software.

Princípios básicos de software de entrada e saída

O subsistema de E/S de um SO é um software bastante complexo devido principalmente à diversidade de periféricos que ele deve tratar. Mesmo dentro de uma mesma “classe de dispositivos”, existe uma grande variedade, como

por exemplo, placas de rede com protocolos, tecnologias e velocidades diferentes, ou ainda discos magnéticos do tipo SCSI, IDE, EIDE, ZIP, CD-ROM, etc. O objetivo primeiro do subsistema de E/S é padronizar ao máximo a forma de acesso aos periféricos. Para atingir esse objetivo, o subsistema de E/S é normalmente organizado em uma estrutura de quatro camadas, onde cada camada fornece funções (serviços) à camada superior. A Figura 5.1 apresenta essa estrutura.

A camada mais inferior é composta pelo hardware dos dispositivos de E/S. A interface que ela apresenta à camada imediatamente superior (*drivers* de dispositivos) é composta pelos mecanismos apresentados na Seção 5.1. Em resumo, o hardware interage com os *drivers* através das interfaces físicas (paralelas ou seriais) e seus controladores e pelo emprego de interrupções e DMA. A seguir, estudaremos o software de E/S.

Drivers de dispositivo

A camada inferior de software ─ *driver*s de dispositivos (*device drivers*) ─ é composta por um conjunto de módulos, cada um responsável por implementar o acesso a um dispositivo específico. O principal objetivo dos *driver*s é "esconder" as diferenças entre os vários dispositivos de E/S, fornecendo à camada superior uma "visão uniforme" desses dispositivos através de uma interface de programação única.

A camada de *drivers* é responsável por implementar as rotinas necessárias ao acesso e à gerência de cada dispositivo. É nesse nível que o software de E/S realiza a programação de registradores internos de controladores que compõem a interface física dos dispositivos e implementa os respectivos tratadores de interrupção. Assim, cada tipo de dispositivo requer um *driver* apropriado. Essa camada fornece uma abstração a mais genérica possível para a camada superior, a de E/S independente do dispositivo.

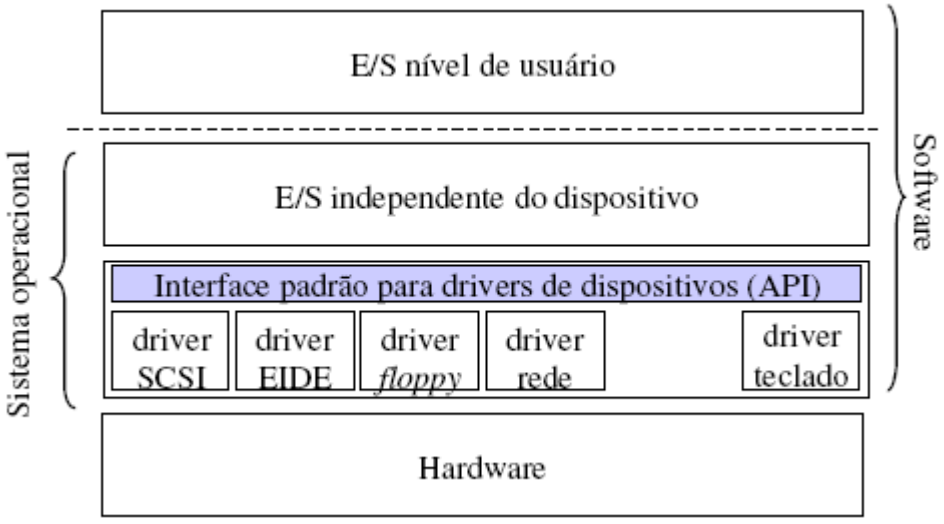


Figura – Estrutura em camadas do subsistema de E/S.

Para fornecer esta "visão uniforme", os dispositivos de E/S são classificados em duas grandes categorias, de acordo com a unidade de transferência de dados, que pode ser bloco ou caractere (byte). Em um **dispositivo orientado a bloco** (*block devices*), o armazenamento de dados e a transferência são realizados através de blocos de tamanho fixo. Tipicamente, o tamanho de um bloco varia entre 512 bytes e 32 kbytes. As unidades de disco são o exemplo

mais comum de dispositivos orientados a bloco. Os **dispositivos orientados a caractere** (*character devices*) realizam as transferências byte a byte, a partir de um fluxo de caracteres, sem necessidade de considerar uma estrutura qualquer. As portas seriais são exemplos de dispositivos orientados a caractere. Essa classificação, entretanto, não é completamente adequada, pois nem todos os dispositivos de E/S podem ser enquadrados em um desses dois grupos. Os temporizadores (relógios) e monitores de vídeo são exemplos de dispositivos que não se enquadram em nenhuma dessas categorias. Existe ainda um outro tipo de dispositivo denominado **pseudo-dispositivo** (*pseudo-devices*) que na realidade não corresponde a nenhum periférico físico. Ele é apenas uma abstração empregada para adicionar funcionalidades ao SO, explorando a interface padronizada já existente para o tratamento de dispositivos. É dessa forma, por exemplo, que o sistema UNIX permite o acesso à memória principal como se fosse um disco (*ramdisk*), ou ainda o emprego do dispositivo nulo (*/dev/null*) para descartar dados.

E/S independente do dispositivo

Pode-se dizer que a camada dos *drivers* é a camada “dependente dos dispositivos”, já que nela estão as funções diretamente ligadas ao hardware dos dispositivos. A camada acima usa apenas a interface abstrata ou virtual (padronizada e mais amigável) provida pelos *drivers* e, como tal, pode ser considerada “independente de dispositivo”. A camada independente de dispositivo implementa funções genéricas (no sentido de valer para qualquer dispositivo) e serviços gerais de E/S, importantes para o funcionamento do sistema como um todo. As funções genéricas são implementadas através de estruturas que representam “classes” de dispositivos e operações associadas. Internamente, essas estruturas possuem ponteiros para descritores que especializam as operações. Por exemplo, pode-se ter uma função genérica **read** cujo primeiro argumento indica o dispositivo a ser usado. Essa operação genérica vai ser mapeada para uma seqüência de operações compatíveis com o dispositivo em questão, pois ler de um teclado é diferente de ler de um disco.

Alguns serviços sob responsabilidade desta camada são:

Escalonamento de E/S: Usado em dispositivos compartilhados por vários processos (por exemplo, discos magnéticos) para melhorar o desempenho dos mesmos.

Buferização: Um exemplo típico de buferização ocorre em protocolos de comunicação; o usuário pode desejar enviar, digamos, 64 Kbytes, mas a interface de rede pode enviar apenas seqüências máximas de 4 Kbytes. Nesse caso, é necessário armazenar a requisição do usuário e enviá-la em blocos de 4 kbytes.

***Cache* de dados:** Consiste em armazenar na memória os blocos de dados freqüentemente acessados. Um exemplo são as *caches* de disco (esse mecanismo será apresentado quando estudarmos sistemas de arquivos).

Alocação de dispositivo: Muitos dispositivos admitem, no máximo, um

usuário por vez. Esse controle é normalmente efetuado através da técnica de ***spooling***, que consiste em seqüencializar os pedidos de acesso e atendê-los um a um. Os pedidos são registrados em uma fila especial (*spool*), a qual é acessada por um processo especial do SO (*daemon*), o qual atende as requisições de E/S. A gerência de impressora é um exemplo clássico do emprego de *spool*.

Direitos de acesso: Nem todos os usuários podem acessar os dispositivos da mesma forma e cabe ao SO garantir essa proteção. O sistema UNIX, por exemplo, emprega os bits de permissão de acesso a arquivos (*rwx* - leitura, escrita e execução) para selecionar o tipo de operação que um determinado usuário, ou grupo de usuários, pode efetuar sobre um dispositivo particular.

Tratamento de erros: O software de E/S deve ser capaz de tratar erros, informando à camada superior o sucesso ou o fracasso de uma operação. Erros transientes – como impossibilidade de transmissão por *overflow* em *buffers* – podem implicar apenas uma nova tentativa e não no término do processo requisitante.

Entrada e saída à nível de usuário

O usuário "vê" os periféricos através dos aplicativos e das linguagens de programação que ele utiliza. No caso de um editor de textos, por exemplo, as operações de leitura e escrita dos registros do arquivo editado ficam invisíveis, sendo feitas automaticamente pelo aplicativo. No caso de uma linguagem de programação, o usuário escreve comandos de E/S de alto nível (por exemplo, a função printf() da linguagem C para realizar saída formatada de dados), os quais são traduzidos para um código que contém chamadas para rotinas de uma biblioteca de E/S. O fabricante do compilador é responsável por fornecer uma biblioteca de E/S para cada sistema em que seu compilador vá ser utilizado. As funções dessa biblioteca contém as chamadas para o SO. As bibliotecas são fornecidas na forma de um módulo objeto (relocável), o qual é ligado com o programa do usuário para compor o executável. É importante notar que as bibliotecas de E/S não fazem parte do SO, pois elas são associadas às linguagens de programação e/ou aos aplicativos de desenvolvimento.

Dispositivos periféricos típicos

Existem atualmente uma grande gama de dispositivos periféricos que possibilitam a comunicação do homem com o computador e entre computadores. A seguir serão abordados apenas os dispositivos mais importantes e/ou comuns em computadores (disco, vídeo, teclado e rede).Na perspectiva do SO, o periférico mais importante é o disco magnético, principalmente o disco rígido. Ele desempenha um papel fundamental em diversos aspectos do SO, servindo para desde o simples armazenamento de dados até a implementação de mecanismos complexos como a memória virtual.

Discos rígidos

Os discos rígidos são dispositivos capazes de armazenar grande volume de dados. O atual estado tecnológico permite a construção de discos magnéticos de vários gigabytes a um baixo custo, sendo encontrados em qualquer computador de uso pessoal. A unidade de disco (ver figura 5.2) contém um conjunto de discos metálicos (aço ou alumínio) superpostos, que giram em torno de um eixo central (*spindle*). A tecnologia atual permite superpor até 8 discos. As duas superfícies de cada disco são recobertas por uma película magnética na qual os dados são gravados. O eixo gira a uma rotação constante (3600 rpm, por exemplo). Os cabeçotes de leitura/gravação (um para cada superfície de disco) realizam movimentos de vai-e-vem e podem se deslocar desde a borda do disco até o centro. Graças ao movimento de rotação dos discos e ao movimento retilíneo dos cabeçotes, toda a superfície de cada disco pode ser alcançada por seu respectivo cabeçote.
Cada superfície é dividida em circunferências concêntricas denominadas **trilhas (*tracks*)**, as quais são divididas radialmente em unidades chamadas **setores (*sectors*)**. Todos os setores tem o mesmo tamanho, o qual varia entre 512 a 4096 bytes (sendo mais comum 512). O setor constitui a unidade mínima de leitura e gravação em um disco. O conjunto de trilhas de todas as superfícies que ficam exatamente à mesma distância do eixo central forma o **cilindro (*cylinder*)**.
A definição das trilhas e setores de um disco chama-se **formatação física** e é um procedimento realizado pelo fabricante. É fácil observar que, sendo divisões radiais do disco, os setores correspondentes às trilhas mais externas são mais longos que os setores de trilhas mais internas. Como, em princípio, os setores armazenam a mesma quantidade de dados, a densidade de gravação nos setores externos é menor que no setores internos.

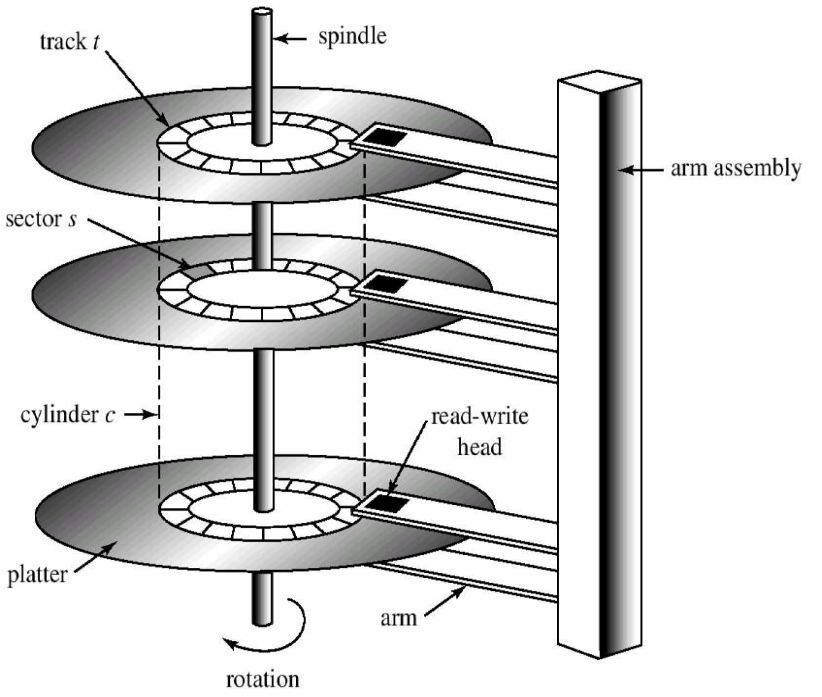


Figura- Organização física do disco magnético.

Para acessar dados no disco, é necessário informar, ao controlador, o cilindro, a superfície e o setor a ser acessado. Esse método de acesso é denominado CHS (*Cylinder, Head, Sector*). Outra maneira consiste em "enxergar" o disco como um conjunto de blocos, formados por um oumais setores. Neste caso, é informado o número do bloco a acessar e este é convertido no endereço físico CHS (cilindro, superfície, setor) por um procedimento que se denomina de LBA (*Linear Block Addressing*). Outros termos bastante comuns associados a discos rígidos são **formatação lógica** e **partições**. Ambos os conceitos estão mais relacionados com o sistema de arquivos do que com o disco propriamente dito. A formatação lógica consiste em gravar informações no disco de forma que arquivos possam ser escritos, lidos e localizados pelo SO. Por sua vez, o conceito de partição está associado à capacidade de dividir logicamente um disco em vários outros discos.

Tempo de acesso

Para realizar um acesso a um disco rígido, é necessário posicionar o cabeçote de leitura/escrita no setor onde o dado será lido ou escrito. Considerando a organização de um disco, esse procedimento de posicionamento implica um certo tempo: **o tempo de acesso**. Esse tempo é formado por três parcelas:
t t t taccess seek latency transfer:

_ ***Seek*** **time (*tseek*):** é o tempo necessário para deslocar os cabeçotes até o cilindro onde está a trilha a ser acessada; a seleção da trilha (cabeçote) é feita eletrônicamente, em tempo zero.

_ **Latency time (*tlatency*):** é o tempo necessário para o cabeçote se posicionar no início do setor a ser lido ou escrito. Esse tempo também é denominado de atraso rotacional (*rotational delay*). _ **Transfer time (*ttransfer*):** é o tempo necessário para realizar a transferência dos dados (leitura ou a escrita dos dados).
O tempo predominante é o tempo de *seek*. Por essa razão, alguns discos antigos possuíam um sistema com cabeçotes fixos, um para cada trilha do disco.

Entrelaçamento
Outro fator relacionado com a redução do tempo de acesso a um disco é o **entrelaçamento** (*interleaving*). É muito comum o acesso a vários setores contíguos em uma trilha do disco. Suponha que se deseja ler os setores 4 e 5 de uma determinada trilha. O SO envia ao controlador de disco o comando para ler o setor 4. Após o *seek* apropriado, o cabeçote passa sobre o setor 4, e a transferência ocorre. Quando o cabeçote sai do setor 4, os dados são transferidos do *buffer* do controlador para a memória, provocando uma interrupção no processador para informar o término da leitura do setor 4. Nesse momento, o processador (via SO) envia um novo comando de leitura, dessa vez para o setor 5. Um novo *seek* não será necessário, pois o cabeçote já se encontra sobre o cilindro desejado. Entretanto, devido à rotação do disco, o cabeçote provavelmente não se encontra mais no início do setor 5. Será necessário esperar que o disco dê uma volta completa (tempo de latência) para

então efetuar a leitura do setor 5. A forma usual para evitar esse problema é realizar um entrelaçamento dos setores (*interleaving*). Essa técnica numera os setores não mais de forma contígua mas sim com um espaço entre eles. O disco 2 da Figura 5.3 possui um fator de entrelaçamento igual a 2. Isso significa que entre o setor *k* e o setor *k1,* existem dois outros setores. Podemos considerar que o disco 1 nessa figura possui um fator de entrelaçamento igual a zero. Voltando ao exemplo no qual os setores 4 e 5 são lidos, mas agora considerando um entrelaçamento de 2, após o cabeçote sair do setor 4, ele passa sobre os setores 15 e 10 antes de chegar no início do setor 5. Desta forma, quando o processador mandar o comando de leitura do setor 5, este não terá passado ainda sob o cabeçote. O melhor fator de entrelaçamento para uma determinado disco depende da velocidade do processador, do barramento, do controlador e da velocidade de rotação do disco.

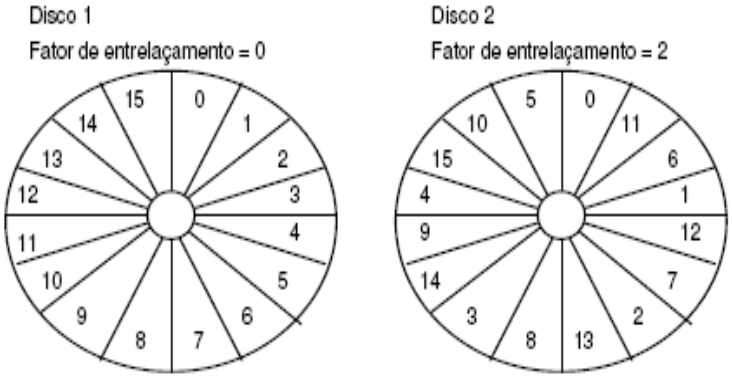


Figura - Trilha com 16 setores e diferentes fatores de entrelaçamento.

Escalonamento de disco

Como se sabe, uma das principais funções do SO é gerenciar os recursos do sistema de forma eficaz. Por outro lado, o disco rígido é um dos principais recursos de um sistema multiprogramado. Uma operação de E/S em disco envolve uma chamada de sistema e a especificação de uma série de parâmetros: tipo de operação (leitura ou escrita), número de bytes a serem transferidos, endereço de memória (*buffer*) e endereço no disco. No caso de um sistema multiprogramado, pode-se ter vários processos realizando "simultaneamente" pedidos desse tipo e sendo bloqueados até que a operação seja realizada. O problema do escalonador do disco consiste em ordenar e atender os pedidos de E/S, de forma a realizar um bom atendimento, buscando minimizar o tempo em que processos permanecem bloqueados.

O tempo necessário a uma operação de E/S em disco tem como principal componente o tempo de *seek*. Nos algoritmos de escalonamento apresentados a seguir, todos eles, com exceção do FCFS (ou FIFO), se preocupam em atender os pedidos de forma a minimizar o tempo médio de *seek*.

_ **FCFS (*First Come First Served*):** É o algoritmo de escalonamento mais simples. As solicitações de acesso são atendidas na ordem em que os pedidos são feitos.

_ **SSTF (*Shortest Seek Time First*):** O próximo pedido a ser atendido é aquele que se refere ao cilindro mais próximo do cilindro atual (isto é, aquele que

envolve a menor movimentação do braço do disco). Novos pedidos são ordenados em relação ao cilindro atual, usando uma lista duplamente encadeada. Como sempre é selecionado o pedido correspondente ao cilindro mais próximo, pode ocorrer postergação indefinida (*starvation*) de um pedido que refere um cilindro distante.

_ **SCAN:** Esse algoritmo é uma variação do SSTF. Ele se diferencia por estipular
um sentido preferencial para o movimento do cabeçote, como por exemplo, do cilindro mais externo para o mais interno. O algoritmo SCAN opera da seguinte maneira: enquanto restam pedidos no sentido corrente, o braço do disco continua se movendo nesse sentido, atendendo os pedidos (correspondentes aos cilindros) mais próximos; se não há mais pedido no sentido corrente (possivelmente porque o fim da superfície foi atingido), então o disco inverte o sentido do braço e se inicia uma varredura no sentido inverso.

O comportamento do algoritmo SCAN é similar ao de um elevador (por isso ele também é denominado **algoritmo do elevador**). Sua vantagem é eliminar a possibilidade de *starvation.*

_ **C-SCAN (*Circular SCAN*)**: Neste algoritmo os pedidos são atendidos em um só sentido da varredura, por exemplo, do cilindro mais externo para o mais interno. O C-SCAN elimina o seguinte defeito do algoritmo SCAN: imediatamente após inverter a varredura, o SCAN privilegia os pedidos correspondentes aos cilindros recém servidos e, por conseqüência, os pedidos dos cilindros do outro extremo da varredura – mais antigos – devem esperar.

Muitos fatores influem no desempenho de um algoritmo de escalonamento de disco, entre eles o número de pedidos (carga), a organização dos arquivos e a estrutura de diretórios do disco. Por essa razão, para facilitar a adequação do algoritmo de escalonamento a um sistema específico, este componente é um módulo a parte do SO.

Para discos que não apresentam carga de trabalho elevada (por exemplo, *CD-ROM* e *ZIP disk*), o algoritmo de escalonamento é do tipo FCFS.

Vídeo

O funcionamento desse dispositivo pode ser melhor entendido considerando as unidades de vídeo antigas. Os primeiros terminais de vídeo eram formados por um teclado e um monitor cuja tela era organizada em uma matriz composta por 40 linhas e 80 colunas, na qual cada elemento correspondia a um caractere. Os caracteres, por sua vez, eram armazenados em uma memória de vídeo, cujas posições correspondiam a elementos dessa matriz. Na realidade, a cada elemento da matriz eram associados dois bytes: um que correspondia ao código ASCII do caractere, e outro ao atributo (fundo invertido, piscando, etc). Devido a essa característica, esse tipo de terminal é denominado de “memória mapeada”. Alguns desses terminais ainda eram ditos inteligentes, pois eram capazes de aceitar seqüências de comandos como movimentar e posicionar o cursor, limpar a tela, executar rolamento de tela (*scroll*). Internamente, o terminal possuia um controlador de vídeo e um controlador de comunicação serial. Embora hoje em dia a realidade seja outra, o princípio continua o mesmo. A diferença dos vídeos atuais está em dois pontos. O primeiro é que o controlador de vídeo é integrado na placa mãe do computador, ou interconectado diretamente no barramento do computador através de uma

interface de vídeo. Isto aumenta consideravelmente a interação entre vídeo, processador e outros dispositivos de E/S. O segundo é a adição de capacidades gráficas aos vídeos. Os vídeos continuam com a mesma estrutura de seus ancestrais, isto é, o vídeo ainda é visto como uma matriz de linhas e de colunas, entretanto essa matriz pode ser acessada de dois modos diferentes: texto ou gráfico. No modo texto, o vídeo é organizado em uma matriz de caracteres e as posições da memória de video correspondem a caracteres e seus atributos. No modo gráfico, a matriz é organizada em *pixels*; cada *pixel* tem associada uma série de posições de memória correspondentes a suas cores. A memória de vídeo determina a resolução (número de linhas e colunas da matriz) e o número de cores de cada *pixel*. A capacidade de resolução e número de cores determina o tipo do controlador (EGA, VGA, SVGA, etc).

O *driver* de terminal (vídeo) continua com a mesma função básica do início da era dos terminais: programar o controlador de vídeo. Essa programação consiste essencialmente em determinar a memória de vídeo disponível, o modo de operação (texto ou gráfico), freqüências para a varredura horizontal e vertical, etc. Os ambientes gráficos X-windows, gnome, KDE, etc., constituem uma aplicação que intercepta e processa as informações provenientes do *driver* de teclado e do mouse. Esses ambientes constróem imagens na memória de vídeo e enviam comandos para o controlador de vídeo de forma a termos a realimentação visual que estamos acostumados a ver.

As placas aceleradoras e 3D, bastante em moda atualmente, também seguem esse mesmo princípio. A diferença está no fato de que o controlador de vídeo é mais potente e realiza por hardware uma série de funções bastante comuns em efeitos de animações e de desenho. No caso específico de uma placa 3D, existe um potente processador gráfico capaz de efetuar operações de visualização como perspectiva, iluminação, etc., bastando apenas, para isso que a aplicação forneça um comando.

Teclado

O teclado é o principal periférico de entrada, utilizado na interação direta dos usuários com o computador. O princípio de operação do teclado é bastante simples: gerar um símbolo para cada tecla pressionada. Mecanicamente, um teclado pode ser visto como uma matriz de *i* linhas e *j* colunas as quais entram em contato quando uma tecla é pressionada. A cada elemento *i,j* da matriz corresponde um caractere (tecla). Quando uma tecla é pressionada, o teclado identifica a linha e a coluna associadas a essa tecla e gera um código denominado *scan code* (código de varredura), o qual é colocado no registrador de dados da interface do teclado. Além disso, gera uma interrupção o que ocasiona a execução do tratador de interrupções do teclado. O tratador lê o registrador de dados para recuperar o *scan code* da tecla pressionada. Com base no *scan code*, o tratador consulta uma tabela interna, substituindo o *scan code* pelo código ASCII correspondente à tecla pressionada. O código ASCII, em seguida, é armazenado em uma região especial da memória (*buffer* de teclado) de onde poderá ser recuperado por chamadas do SO. A leitura de caracteres é disponibilizada ao usuário final através de rotinas de biblioteca do tipo getc(), da linguagem C.

Alguns detalhes foram omitidos na descrição acima, pois nosso objetivo é dar

apenas uma visão geral do funcionamento do teclado. Na verdade, para cada tecla, são geradas duas interrupções (com *scan code* diferentes), uma para indicar o pressionar da tecla e outra para sinalizar que a tecla foi liberada. Lembre que a tecla *shift*, por exemplo, fica pressionada enquanto se digita uma letra maiúscula. Nós, usuários, queremos visualizar os caracteres que digitamos na tela de nosso computador. Esse procedimento de ler dados do teclado e escrevê-los na tela denominase **ecoamento**. Outra característica bastante comum na utilização de computadores é a possibilidade de "abrir" várias janelas. Nesse caso, os caracteres digitados devem ser direcionados à janela correta. Esse "direcionamento" é feito através do conceito de "janela ativa" e da interação entre o *driver* de teclado e o *driver* de vídeo sobre o *buffer* de teclado.

Rede

Uma rede pode ser genericamente definida como um conjunto de computadores interconectados de forma a compartilhar recursos comuns: discos, impressoras, arquivos, etc. A forma mais simples de rede é uma rede ponto a ponto, na qual dois computadores podem ser interligados diretamente através de suas interfaces paralelas e seriais padrões.

Entretanto, o que se convenciona denominar de rede supõe o emprego de um hardware especial – a interface de rede – que provê a interconexão de várias máquinas segundo uma tecnologia específica. A tecnologia empregada pelo hardware da interface, ethernet ou ATM, por exemplo, determina como fisicamente os dados serão transmitidos, a velocidade de transmissão, a capacidade de transmitir e receber ao mesmo tempo (*full duplex*), entre outras. Independente da tecnologia e dos aspectos físicos ligados à transmissão/recepção, o software de gerenciamento deve tratar de problemas similares.

Uma placa de rede é constituída por um hardware que transforma os sinais digitais em sinais analógicos segundo sua tecnologia. Ela também possui internamente uma capacidade de memória (*buffers*) na qual os dados a serem transmitidos, ou recebidos, são armazenados. Algumas placas de rede possuem um processador dedicado a essa função (o controlador de rede). A placa de rede trabalha sob um certo aspecto orientada a eventos. Um evento é o final da transmissão. A ocorrência desse evento é interpretada como disponibilidade para nova transmissão. Outro evento é a recepção de uma mensagem. Nesse caso, a mensagem deve ser lida. Essa descrição, entretanto, é extremamente simplista. O software de rede é bastante complexo e envolve muitas abstrações; por isso, normalmente ele é organizado em camadas (você já deve ter ouvido falar no modelo OSI/ISO).

Para apresentar um pouco o grau de dificuldade e os problemas a serem gerenciados por um software de rede, vamos examinar o caminho seguido por uma mensagem. Vamos considerar que um processo A quer enviar uma mensagem para um processo B em uma máquina remota. A quantidade de bytes dessa mensagem pode ser maior que a quantidade máxima permitida fisicamente para transmissão. Essa quantidade depende da tecnologia de rede empregada. Cada tipo de rede suporta o que se denomina de MTU (*Maximum Transfer Unit*). Se nossa mensagem é maior que o MTU, ela deve ser dividida em pedaços (pacotes) de tamanho máximo igual a MTU. A primeira tarefa do

software do lado remetente é realizar essa “quebra” da mensagem original. Esse processo denomina-se **fragmentação**. Ao final da transmissão de cada pacote, uma interrupção é gerada pela placa de rede para sinalizar esse evento e indicar que ela está pronta para enviar outro pacote.

No outro extremo, a máquina do processo B, recebe os pacotes. Para cada pacote recebido é gerada uma interrupção. Esta interrupção tem por objetivo transferir este pacote para um *buffer* maior e reconstruir com os pacotes a mensagem original. Aqui surge outro problema. Como o *driver* de rede identifica que estes dados são para o processo B e não para um outro processo C? Pior ainda, como ele não mistura estes pacotes? Uma solução é criar canais lógicos entre os processos. Desta forma o software de rede antes de começar a enviar/receber dados deve estabelecer uma conexão (canal lógico) entre os dois processos que desejam comunicar. Este canal lógico identificará então os pares. O *driver* de rede – estando consciente dos canais – poderá redirecionar os pacotes recebidos para um, ou para outro canal.

Esse cenário, entretanto, é simplificado, pois ele não considera a ocorrência de erros. Os pacotes podem ser perdidos, ou por erros de transmissão no meio físico, ou por falta de espaço no *buffer* do destinatário. Pacotes podem ainda ser invertidos, isto é, o pacote *i+1* pode chegar antes do pacote *i*. Isto pode ocorrer quando, devido a um erro, um pacote *i* é retransmitido, enquanto o *i+1* foi recebido corretamente, ou ainda quando o pacote *i+1* utiliza um caminho mais curto que o pacote *i* (roteamento). O destinatário deve então ser responsável por contar e por sinalizar ao remetente a perda de pacotes. Ele deve ainda ser capaz de ordenar os pacotes recebidos.

Todos esses procedimentos e muitos outros são gerenciados pelo que se denomina de protocolo. Um exemplo bastante conhecido de protocolos é a família TCP/IP, pois toda a Internet está baseada nessa família. Os problemas mencionados acima, entre outros, são tratados pela implementação de protocolos, os quais são organizados em níveis (camadas). No nível mais baixo, estão as placas de rede; no mais alto, a aplicação do usuário.

Exercícios

1) Mostre como é a técnica de *interleaving* (entrelaçamento) utilizada em discos e para que ela serve.

2) Descreva o funcionamento da otimização de *seek* em acesso a disco quando o algoritmo SSTF (*shortest seek time first* - menor *seek* primeiro) é utilizado.

4) Descreva o funcionamento da otimização de *seek* em acesso a disco quando o algoritmo SCAN (elevador) é utilizado.

5) Considere que um disco tenha 100 cilindros (numerados de 0 a 99), que o braço do disco esteja posicionado no cilindro 0 (mais externo), que o tempo atual seja 0, que uma operação de E/S demore (**10+d**) u.t. (unidades de tempo), onde **d** é a distância percorrida pelo braço1 e que ocorra a seqüência de pedidos de acesso mostrada abaixo.

(a) Preencha a segunda coluna com o tempo no qual será atendido cada pedido, considerando o algoritmo SSTF;

(b) Preencha a terceira coluna com o tempo no qual será atendido cada pedido, considerando o algoritmo SCAN.

Pedido		Atendimento SSTF	Atendimento SCAN
tempo	cilindro		
0	30	0 – 40	0 – 40
10	20		
15	50		
25	40		
30	35		
35	30	40 – 50	
45	10		
55	70		

2.5 – Sistemas de arquivos

A forma como a controladora vê os dados armazenados nos discos magnéticos pode ser bem diferente da forma como vê o sistema operacional. Enquanto a controladora enxerga as trilhas, setores e cilindros e se esforça para localizá-las nos discos magnéticos usando as marcações servo, o sistema operacional enxerga apenas uma longa lista de endereços, chamados de clusters ou blocos. Quando ele precisa de um determinado arquivo, ele não se preocupa em tentar descobrir em qual trilha e setor ele está armazenado. Ele apenas envia o endereço do bloco que deve ser lido e a controladora se encarrega do restante.

O fato da controladora "esconder" as informações sobre a organização interna dos discos, é o que faz com que os sistemas operacionais sejam compatíveis com todos os HDs do mercado, sem que seja necessário instalar drivers completos para cada um. Quando acontece de uma versão antiga do Windows, ou de alguma distribuição Linux não detectar seu HD durante a instalação, quase sempre o problema é causado pela falta de drivers para a interface IDE ou a controladora SATA do chipset da placa mãe, e não para o HD em si. A primeira versão do Windows XP, por exemplo, não oferecia suporte nativo à maioria das controladoras SATA, de forma que você precisava fornecer um disquete com drivers durante a instalação.
Formatação física

Originalmente, os discos magnéticos do HD são um terreno inexplorado, uma mata virgem sem qualquer organização. Para que os dados possam ser armazenados e lidos de forma organizada, é necessário que o HD seja previamente formatado.

Em primeiro lugar, temos a formatação física, onde os discos são divididos em trilhas, setores e cilindro e são gravadas as marcações servo, que permitem que a placa lógica posicione corretamente as cabeças de leitura.

Nos HDs atuais, a formatação física é feita em fábrica, durante a fabricação dos discos. O processo envolve o uso de máquinas especiais e, apenas para garantir, restrições são adicionadas no firmware do drive, para que a placa lógica seja realmente impedida de fazer qualquer modificação nas áreas reservadas. Graças a isso, é impossível reformatar fisicamente um drive atual, independentemente do software usado.

No caso dos drives "pré-ATA", como os antigos ST-506 e ST-412 a história era diferente. Eles precisavam ser periodicamente reformatados através do setup, pois quando lida pela cabeça de leitura, a mídia magnética dos discos esquentava e se expandia, esfriando e contraindo-se logo em seguida. Esta expansão e contração da superfície do disco, acabava por alterar a posição das trilhas, causando desalinhamento e dificultando a leitura dos dados pela cabeça magnética. Era necessária então uma nova formatação física, para que as trilhas, setores e cilindros, voltassem às suas posições iniciais.

No caso dos discos atuais, este processo não é mais necessário, pois as mídias são muito mais confiáveis e a placa controladora pode compensar eventuais desvios rapidamente, simplesmente calibrando o movimento do braço de leitura.

Formatação lógica

Em seguida, temos a formatação lógica, que adiciona as estruturas utilizadas pelo sistema operacional. Ao contrário da formatação física, ela é feita via software e pode ser refeita quantas vezes você quiser. O único problema é que, ao reformatar o HD, você perde o acesso aos dados armazenados, embora ainda seja possível recuperá-los usando as ferramentas apropriadas, como veremos mais adiante.

Chegamos então ao sistema de arquivos, que pode ser definido como o conjunto de estruturas lógicas que permitem ao sistema operacional organizar e otimizar o acesso ao HD. Conforme cresce a capacidade dos discos e aumenta o volume de arquivos e acessos, esta tarefa torna-se mais e mais complicada, exigindo o uso de sistemas de arquivos cada vez mais complexos e robustos.

Existem diversos sistemas de arquivos diferentes, que vão desde sistemas simples como o FAT16, que utilizamos em cartões de memória, até sistemas como o NTFS, EXT3 e ReiserFS, que incorporam recursos muito mais avançados.

A formatação do HD é feita em duas etapas. A primeira é o particionamento, onde você define em quantas partições o HD será dividido e o tamanho de cada uma. Mesmo que você não pretenda instalar dois sistemas em dual boot, é sempre interessante dividir o HD em duas partições, uma menor, para o sistema operacional, e outra maior, englobando o restante do disco para armazenar seus arquivos. Com isso, você pode reinstalar o sistema quantas vezes precisar, sem o risco de perder junto todos os seus arquivos.

Podemos ter um total de 4 partições primárias ou três partições primárias e mais uma partição estendida, que pode englobar até 255 partições lógicas. É justamente a partição lógica que permite a nós dividir o HD em mais de 4 partições.

Esta limitação das 4 partições primárias é uma limitação que existe desde o primeiro PC, lançado em 1981. Os projetistas que escreveram o BIOS para ele

precisavam economizar memória e chegaram à conclusão que 2 bits (4 combinações) para o endereço das partições seriam suficientes, pois na época os HDs mais vendidos tinham apenas 5 MB e só existia um sistema operacional para PCs (o MS-DOS), de forma que era raro alguém precisar criar mais de uma partição. As coisas mudaram "um pouco" de lá pra cá, mas infelizmente esta limitação continua até os dias de hoje ;).

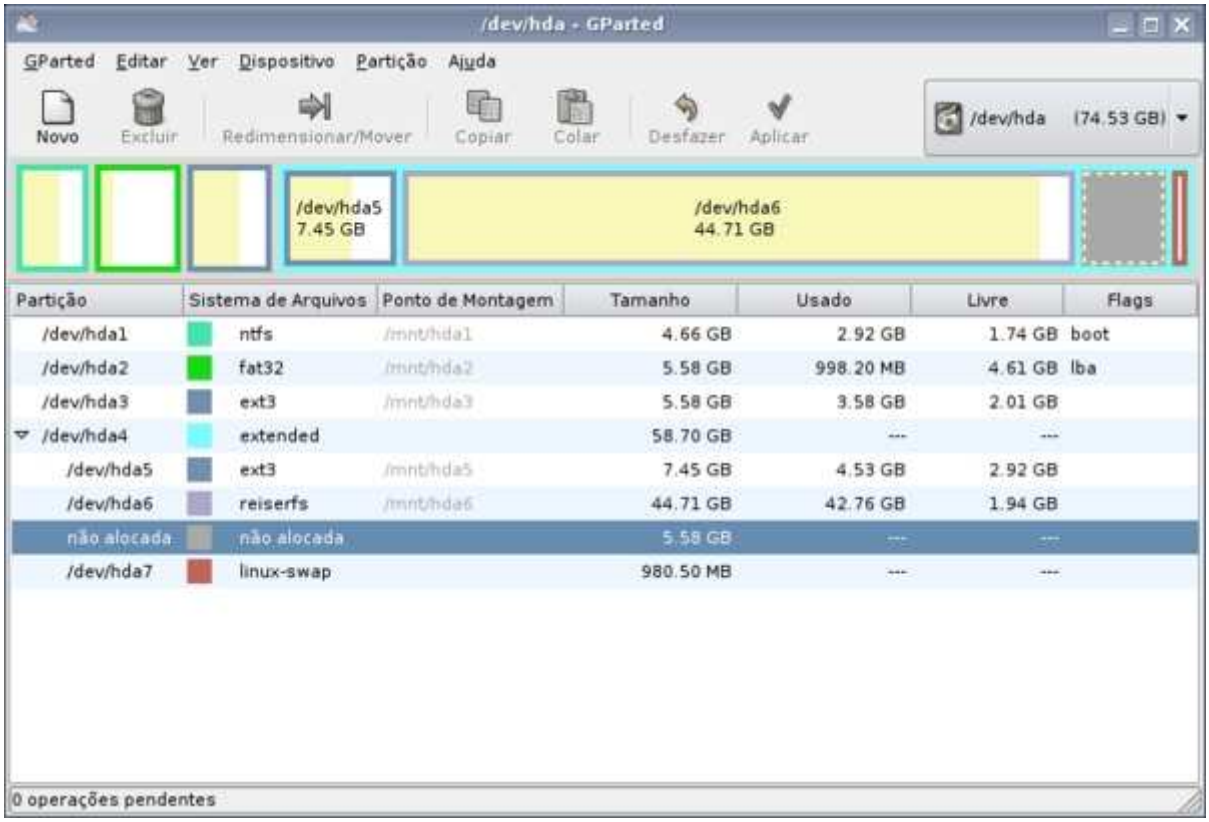
Para amenizar o problema, foi adicionada a possibilidade de criar partições lógicas. Ao invés de criar 4 partições primárias e ficar sem endereços para criar novas partições, você cria uma "partição estendida", que é uma espécie de container, que permite criar mais partições. A partição estendida contém uma área extra de endereçamento, que permite endereçar as 255 partições lógicas. É possível criar até 4 partições estendidas, de forma que (em teoria) é possível dividir o HD em até 1020 partições.

Digamos que você queira particionar um HD de 160 GB para instalar Windows e Linux em dual boot, deixando uma partição de 20 GB para o Windows, uma partição de 20 GB para o Linux, uma partição de 1 GB para swap (do Linux) e uma partição maior, englobando os 119 GB restantes para guardar seus arquivos.

Como precisamos de 4 partições no total, seria possível criar diretamente 4 partições primárias, mas neste caso você ficaria sem endereços e perderia a possibilidade de criar novas partições mais tarde, caso resolvesse testar uma outra distribuição, por exemplo.

Ao invés disso, você poderia começar criando a partição de 20 GB do Windows como primária (é sempre recomendável instalar o Windows na primeira partição do HD e em uma partição primária, devido às particularidades do sistema) e em seguida criar uma partição estendida, englobando todo o resto do espaço, criando as demais partições como partições lógicas dentro dela.

Este é um screenshot do Gparted, que mostra um HD dividido em várias partições. Veja que a quarta partição está marcada como "extended", ou seja, como partição extendida. Ela não armazena dados, nem ocupa um espaço considerável no disco, mas permitiu que fossem criadas as partições de 5 a 7. Veja que existe também um trecho marcado como "não alocada", ou seja, espaço vago onde é possível criar mais uma partição:

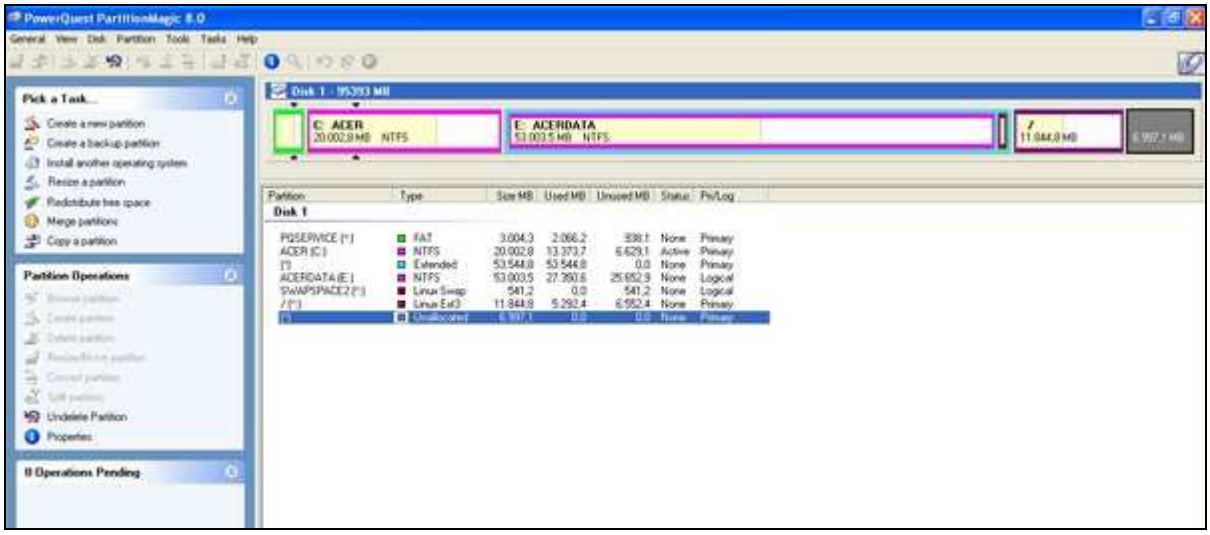


Do ponto de vista do sistema operacional, cada partição é uma unidade separada, quase como se houvessem dois ou três discos rígidos instalados na máquina. Cada partição possui seu próprio diretório raiz e sua própria FAT. As informações sobre o número de partições, sua localização no disco, e o espaço ocupado por cada uma, são armazenadas na tabela de partição, que compartilha o primeiro setor do disco com o setor de boot.

Você pode particionar o HD usando o próprio assistente mostrado durante a instalação do Windows XP ou Vista, dos particionadores mostrados durante as instalação de várias distribuições Linux e também de programas avulsos, como o Partition Magic (no Windows) ou o Gparted (no Linux), que você pode usar dando boot através de uma distribuição live-CD que o traga pré-instalado.

Tanto o PartitionMagic são particionadores gráficos fáceis de usar. O espaço disponível é mostrado na forma de uma barra na parte superior da tela, que vai sendo dividida em retângulos menores, conforme vai criando as partições. A cor de cada partição representa o sistema de arquivos usado e os espaços não particionados do disco aparecem em cinza. Além de criar e deletar partições, os dois programas também oferecem opções adicionais, como redimensionar partições (sem perder os dados), muito útil quando você já tem um sistema operacional instalado e precisa liberar espaço para instalar um segundo sistema em dual boot.

Este é um screenshot do PartitionMagic. Veja que a interface é muito similar à do Gparted, que mostrei a pouco:



Em seguida, temos a formatação propriamente dita, onde as estruturas do sistema de arquivos são finalmente gravadas na partição. Na maioria dos casos, o próprio programa de particionamento se encarrega de formatar a partição usando o sistema de arquivos escolhido, mas em outros temos dois programas diferentes, como no caso do fdisk e do format, usados no Windows 98.

No mundo Windows, temos apenas três sistemas de arquivos: FAT16, FAT32 e NTFS. O FAT16 é o mais antigo, usado desde os tempos do MS-DOS, enquanto o NTFS é o mais complexo e atual. Apesar disso, temos uma variedade muito grande de sistemas de arquivos diferentes no Linux (e outros sistemas Unix), que incluem o EXT2, EXT3, ReiserFS, XFS, JFS e muitos outros. Para quem usa apenas o Windows, estes sistemas podem parecer exóticos, mas eles são velhos conhecidos de quem trabalha com servidores, já que neles o Linux é que é o sistema mais popular.

FAT16 E FAT32

O FAT16 é uma espécie de “pau pra toda obra”, já que é compatível com praticamente todos os sistemas operacionais e também dispositivos como câmeras, palmtops, celulares e MP3players. Ele é o sistema de arquivos usado por padrão nos cartões SD e também nos pendrives de até 2 GB. Só recentemente os cartões passaram a utilizar FAT32, com a introdução do padrão SDHC.

No sistema FAT, o HD é dividido em clusters, que são a menor parcela do HD vista pelo sistema operacional. Cada cluster possui um endereço único, que permite ao sistema localizar os arquivos armazenados. Um grande arquivo pode ser dividido em vários clusters, mas não é possível que dois arquivos pequenos sejam gravados dentro do mesmo cluster. Cada cluster pode ser composto por de 1 a 64 setores (ou seja, de 512 bytes a 32 KB), de acordo com o tamanho da partição.

A principal limitação é que, como o nome sugere, o FAT16 usa endereços de 16 bits para endereçar os clusters dentro da partição, permitindo um máximo

de 65536 clusters, que não podem ser maiores que 32 KB. Isso resulta num limite de 2 GB para as partições criadas.

No caso de HDs (e também pendrives ou cartões) maiores que 2 GB, é possível criar várias partições de 2 GB cada uma, até utilizar todo o espaço disponível. Esta pode ser uma solução no caso de dispositivos com 4 ou 5 GB, por exemplo, mas naturalmente não é uma opção realística no caso de um HD de 60 GB, por exemplo, onde seria necessário criar 30 partições!

Numa partição de 2 GB, cada cluster possui 32 KB, o que acaba resultando num grande desperdício de espaço ao gravar uma grande quantidade de arquivos pequenos. Imagine que gravássemos 10.000 arquivos de texto, cada um com apenas 300 bytes. Como um cluster não pode conter mais do que um arquivo, cada arquivo iria ocupar um cluster inteiro, ou seja, 32 kbytes! No total, os 10.000 arquivos ocupariam um total de 10.000 clusters, ou seja, um total de 320 MB!

O tamanho dos clusters varia de acordo com o tamanho da partição. Quanto maior o tamanho da partição, maior o tamanho dos clusters:

Tamanho da Partição	Tamanho dos Clusters usando FAT16
Entre 1 e 2 GB	32 KB
Menos que 1 GB	16 KB
Menos que 512 Mb	8 KB
Menos que 256 Mb	4 KB

Como em toda regra, existe uma excessão. O Windows NT permitia criar partições FAT de até 4 GB usando clusters de 64 KB, mas este foi um recurso pouco usado, devido ao desperdício de espaço.

A versão original do Windows 95 suportava apenas o FAT16, obrigando quem possuía HDs maiores que 2 GB a dividi-los em duas ou mais partições e a lidar com o desperdício de espaço causado pelos clusters de 32 KB.

A solução foi a criação do sistema FAT32, que foi incorporado no Windows 95 OSR/2 e continuou sendo usado nas versões seguintes.

A principal evolução foi o uso de endereços de 32 bits para o endereçamento dos clusters, o que possibilita a criação de partições muito maiores, de até 2 terabytes. Isto foi possível por que o Windows 95 era um sistema de 32 bits, ao contrário do MS-DOS e do Windows 3.1, que eram sistemas de 16 bits.

A princípio, o uso de clusters de 32 bits permitiriam o uso de clusters de 4 KB mesmo em partições muito grandes, mas por questões de desempenho, ficou estabelecido que por default os clusters de 4 KB seriam usados apenas em partições de até 8 GB. Acima disto, o tamanho dos clusters varia de acordo com o tamanho da partição:

Tamanho da partição	Tamanho do cluster
Menor do que 8GB	4 KB
De 8 GB a 16 GB	8 KB
De 16 BG a 32 GB	16 KB
Maior do que 32 GB	32 KB

Usando clusters de 4 KB, os 10.000 arquivos do exemplo anterior ocupariam apenas 40 MB, uma economia considerável. De fato, ao converter uma partição FAT16 para FAT32, é normal conseguir de 10 a 20% de redução no espaço ocupado, devido à redução do espaço desperdiçado.

O Windows 98 inclui um conversor, que permite converter partições FAT16 para FAT32 sem perda de dados. Atualmente ele não possui muita utilidade, já que o FAT16 é raramente usado fora dos pendrives e cartões de memória, mas de qualquer forma, caso precise dele, o ícone está disponível no Iniciar > Ferramentas de Sistema.

A grande limitação do sistema FAT32 está relacionado ao tamanho máximo dos arquivos. Mesmo usando uma grande partição, não é possível armazenar arquivos com mais de **4 GB**, o que é um grande problema para quem trabalha com arquivos grandes, como vídeos em formato RAW (sem compressão). Não é possível sequer armazenar um ISO de DVD, já que a cópia ou transferência será sempre abortada depois de transferidos os primeiros 4 GB.

Não existe qualquer sinal de que futuras versões do sistema de arquivos derrubarão esta limitação, já que a Microsoft vem recomendando o uso do NTFS desde a primeira versão do Windows XP, de forma que a melhor opção, para quem usa Windows, é seguir a recomendação e migrar para ele.

Outra limitação é que o particionador usado durante a instalação do Windows XP se recusa a formatar partições FAT 32 maiores do que 32 GB. Este é um limite do software e não do sistema de arquivos em si. A solução para criar partições FAT maiores é utilizar o Partition Magic, Gparted ou outro particionador para criar a partição e em seguida apenas instalar o sistema na partição criada.

Uma curiosidade é que, antes do FAT16, existiu o FAT12, um sistema ainda mais primitivo, utilizado em disquetes e também nas primeiras versões do MS-DOS. Nele, são usados endereços de apenas 12 bits para endereçar os clusters, permitindo um total de 4096 clusters de até 4 KB, o que permitia partições de até 16 MB.

Em 1981, quando o IBM PC foi lançado, 16 MB parecia ser uma capacidade satisfatória, já que naquela época os discos rígidos tinham apenas 5 ou 10 MB. Claro que, em se tratando de informática, por maior que seja um limite, ele jamais será suficiente por muito tempo. Um excelente exemplo é a célebre frase “Por que alguém iria precisar de mais de 640 KB de memória RAM?” dita por Bill Gates numa entrevista, no início da década de 80. Logo começaram a ser usados discos de 40, 80 ou 120 MB, obrigando a Microsoft a criar a FAT 16, e incluí-la na versão 4.0 do MS-DOS.

Apesar de obsoleto, o FAT12 ainda continua vivo até os dias de hoje, fazendo compania para outro fantasma da informática: os disquetes. Por ser mais simples, o FAT12 é o sistema padrão para a formatação dos disquetes de 1.44, onde são usados clusters de apenas 512 bytes.

Estruturas Lógicas

Todos os vários sistemas de arquivos são constituídos de um conjunto de estruturas lógicas, que permitem ao sistema operacional organizar os dados gravados e acessá-los com a maior velocidade e confiabilidade possíveis.

Tudo começa com o **setor de boot**, que é lido pelo BIOS da placa mãe no início do boot, logo após a contagem de memória e outros procedimentos executados durante o POST.

O setor de boot, também chamado de MBR ou trilha zero, contém dois componentes essenciais. O primeiro é um bootstrap, o software responsável por iniciar o carregamento do sistema operacional. Tipicamente, é utilizado um gerenciador de boot, como o NTLDR (usado pelo Windows XP) ou o Grub (usado pela maior parte das distribuições Linux). A função do gerenciador de boot é mostrar uma lista com os sistemas operacionais instalados no início do boot e carregar o sistema escolhido.

O bootstrap ocupa os primeiros 446 bytes do MBR. Os 66 bytes restantes são usados para armazenar a **tabela de partições**, que guarda informações sobre onde cada partição começa e termina. Alguns vírus, além de acidentes em geral, podem danificar os dados armazenados na tabela de partição, fazendo com que pareça que o HD foi formatado. Mas, na maioria dos casos, os dados continuam lá, intactos, e podem ser recuperados.

Depois que o disco rígido foi formatado e dividido em clusters, mais alguns setores são reservados para guardar a **FAT** (“file allocation table” ou “tabela de alocação de arquivos”). A função da FAT é servir como um índice, armazenando informações sobre cada cluster do disco. Através da FAT, o sistema sabe se uma determinada área do disco está ocupada ou livre, e pode localizar qualquer arquivo armazenado.

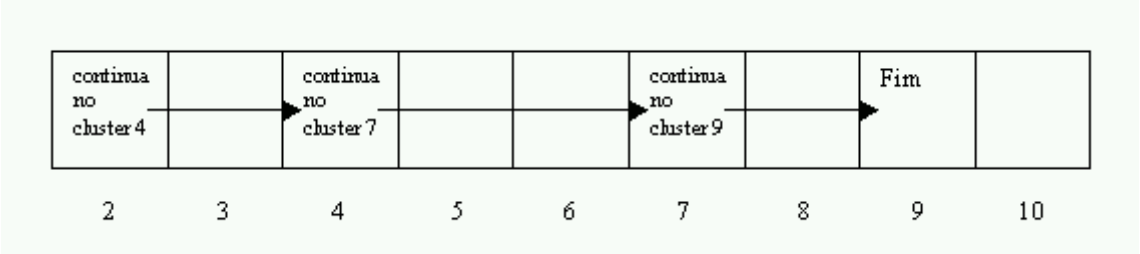
Cada vez que um novo arquivo é gravado ou apagado, o sistema operacional altera a FAT, mantendo-a sempre atualizada. A FAT é tão importante que, além da tabela principal, é armazenada também uma cópia de segurança, que é usada sempre que a tabela principal é danificada de alguma maneira.

Todos os demais sistemas de arquivos utilizam algum tipo de índice, similar à FAT. Quando o o HD é reformatado, este índice é apagado e substituído por uma tabela em branco. Apesar disso, os arquivos continuam gravados nas mesmas posições, embora inacessíveis. Enquanto eles não forem realmente subscritos por outros, é possível recuperá-los usando um programa de recuperação de dados, como veremos em detalhes mais adiante.

Em seguida, temos o **diretório raiz**. Se fôssemos comparar um disco rígido, formatado em FAT16 ou FAT32 com um livro, as páginas seriam os clusters, a FAT serviria como as legendas e numeração das páginas, enquanto o diretório raiz seria o índice, com o nome de cada capítulo e a página onde ele começa.

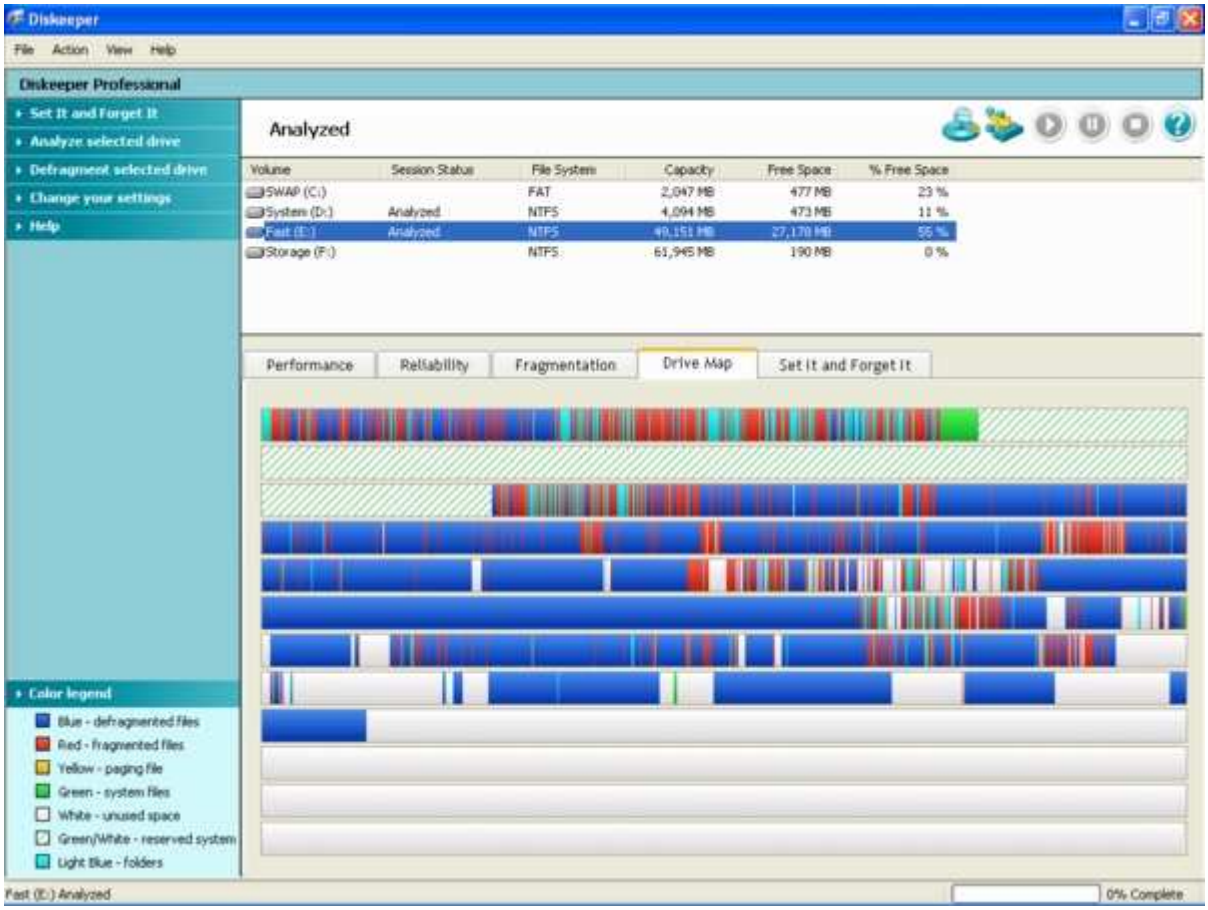
O diretório raiz ocupa mais alguns setores no disco, logo após os setores ocupados pela FAT. Cada arquivo ou diretório do disco rígido possui uma entrada no diretório raiz, com o nome do arquivo, a extensão, a data de quando foi criado ou quando foi feita a última modificação, o tamanho em bytes e o número do cluster onde o arquivo começa.

Um arquivo pequeno pode ser armazenado em um único cluster, enquanto um arquivo grande é “quebrado” e armazenado ocupando vários clusters. Neste caso, haverá no final de cada cluster uma marcação indicando o próximo cluster ocupado pelo arquivo. No último cluster ocupado, temos um código que marca o fim do arquivo.



Quando um arquivo é deletado, simplesmente é removida sua entrada no diretório raiz, fazendo com que os clusters ocupados por ele pareçam vagos para o sistema operacional. Ao gravar um novo arquivo no disco, o sistema simplesmente procura o primeiro setor livre, continuando a gravá-lo nos setores livres seguintes, mesmo que estejam muito distantes uns dos outros. Surge então o problema da fragmentação, que reduz consideravelmente a a velocidade de acesso, já que dados espalhados, significam mais movimentos da cabeça de leitura.

Ao contrário de outros sistemas de arquivos mais modernos, o sistema FAT não possui nenhum mecanismo que impeça, ou pelo menos diminua a fragmentação, daí a necessidade de rodar o defrag ou outro programa desfragmentador periodicamente. A função deles é mover os arquivos, de forma que eles fiquem gravados em clusters seqüenciais.



Uma curiosidade é que a fragmentação é um problema apenas nos HDs, já que eles trabalham com tempos de acesso muito altos. Nos cartões de memória, o tempo de acesso é comparativamente muito baixo, de forma que a fragmentação possui um impacto muito pequeno sobre a performance.

Continuando, além do nome, cada arquivo recebe também uma **extensão** de até três caracteres, como “EXE”, “DOC”, etc. Através da extensão, o sistema operacional sabe que um determinado arquivo deve ser executado ou aberto usando o Word, por exemplo.

A extensão não tem nenhuma influencia sobre o arquivo, apenas determina como ele será visto pelo sistema operacional. Se você abrir o Notepad, escrever um texto qualquer e salvá-lo como “carta.exe”, por exemplo, conseguirá abrir e editar este arquivo sem problemas, mas se você chamar o arquivo clicando duas vezes sobre ele dentro do Windows Explorer, o sistema operacional verá a extensão “EXE” e tentará executar o arquivo, ao invés de tentar abri-lo usando o Notepad, como faria caso o arquivo tivesse a extensão “TXT”.

Inicialmente, as extensões de arquivo eram utilizadas apenas no Windows, enquanto os sistemas Unix em geral se baseavam no conteúdo do arquivo. Mas, recentemente isso vem mudando, com o KDE e o Gnome utilizando associações de arquivos também baseadas nas extensões.

Depois da extensão, existe mais um byte reservado para o atributo do arquivo,

que pode ser “somente leitura”, “oculto”, “sistema”, “volume label”, “diretório” ou “arquivo”. O atributo permite instruir o sistema operacional e demais aplicativos sobre como lidar com o arquivo. Os atributos suportados mudam de acordo com o sistema de arquivos. Como veremos adiante, o NTFS suporta um conjunto de atributos que não existem no sistema FAT. O mesmo pode ser dito em relação aos sistemas de arquivo usados no Linux.

O atributo **“somente leitura”** indica que o arquivo não deve ser modificado ou deletado. Se você tentar deletar ou modificar um arquivo somente leitura pelo DOS, receberá a mensagem “Access Denied”. Tentando apagar o arquivo através do Windows Explorer você receberá um aviso explicando que o arquivo é somente para leitura, perguntando se você tem certeza que deseja deletá-lo. O atributo **“sistema”** possui uma função parecida, indicando apenas que, além de ser oculto, o arquivo é utilizado pelo sistema operacional.

Por padrão, o Windows XP não mostra arquivos marcados com o atributo “sistema”, nem os arquivos ocultos. É necessário configurar o Windows Explorer para exibi-los, marcando a opção marcar a opção “Mostrar todos os arquivos” no menu Exibir/Opções. Outra opção é usar o comando “**DIR /AH**” através da linha de comando.

Para nomear um disquete ou a uma partição de um disco rígido, usamos o atributo **“volume label”**. O nome dado é armazenado em uma área reservada do diretório raiz.

De todos os atributos, o mais importante é o atributo de **“diretório”**, pois ele permite a existência de subpastas. As pastas, mesmo quando vazias, são vistas pelo sistema operacional como arquivos. Dentro deste arquivo ficam armazenadas informações sobre o nome da pasta, atributos como somente leitura, oculto, etc., a posição da pasta na árvore de diretórios (C:\Windows\System, por exemplo) e informações sobre quais arquivos ou subpastas estão guardados dentro dela, assim como a localização destes arquivos no disco.

Como o diretório raiz ocupa (no sistema FAT) um espaço equivalente a apenas 16 KB no disco rígido (32 setores), podemos ter apenas 512 entradas de arquivos ou diretórios. Cada sub-pasta funciona mais ou menos como um novo diretório raiz, permitindo que tenhamos mais arquivos no disco. Como uma pasta (que nada mais é do que um arquivo, marcado com o atributo especial) pode ocupar o espaço que quiser, não temos a limitação de 512 arquivos, como no diretório raiz.

Qualquer arquivo com o atributo “diretório”, passa a ser visto pelo sistema operacional como uma pasta, mas a tentativa de transformar um arquivo qualquer em pasta não daria certo, pois apesar de em essência as pastas também serem arquivos, elas possuem um formato específico.

Uma curiosidade sobre as subpastas é que elas só passaram a ser suportadas a partir da versão 2.0 do DOS. Os usuários do DOS 1.0 tiveram que conviver durante algum tempo com um sistema que permitia armazenar arquivos

apenas no diretório raiz, com a conseqüente limitação de 512 arquivos no HD.

Finalizando, o atributo “arquivo” indica um arquivo que raramente é modificado, ou é uma cópia de backup de algum arquivo importante. Muitos programas de backup verificam este atributo quando fazem um backup incremental (quando são salvos apenas os arquivos que foram alterados desde o último backup). Neste caso, o programa de backup retira o atributo após salvar o arquivo. Ao ser alterado por algum outro programa, o arquivo novamente recebe o atributo, permitindo ao programa de backup saber quais arquivos foram modificados.

Para alterar os atributos de um arquivo através do Windows Explorer, basta clicar sobre ele com o botão direito do mouse e abrir a janela de propriedades. Também é possível alterá-los via linha de comando, usando o comando ATTRIB.

Concluindo, temos o **VFAT**, uma extensão incluída a partir do Windows 95. Inicialmente, o sistema FAT possuía uma grave limitação quanto ao tamanho dos nomes de arquivos, que não podiam ter mais que 11 caracteres, sendo 8 para o nome do arquivo e mais 3 para a extensão, como em “formular.doc”.

O limite de apenas 8 caracteres era um grande inconveniente para os usuários do MS-DOS. O “Boletim da 8º reunião anual de diretoria”, por exemplo, teria de ser gravado na forma de algo como “8reandir.doc”.

Através do VFAT, arquivos com nomes longos são gravados no diretório raiz respeitando o formato 8.3 (oito letras e uma extensão de até 3 caracteres), sendo o nome verdadeiro armazenado numa área reservada. Se tivéssemos dois arquivos, chamados de “Reunião anual de 1998” e “Reunião anual de 1999”, por exemplo, teríamos gravados no diretório raiz “Reunia~1” e “Reunia~2”. Se o disco fosse lido a partir do DOS, o sistema leria apenas este nome simplificado. Lendo o disco através do Windows, é possível acessar as áreas ocultas do VFAT e ver os nomes completos dos arquivos. Isso permitiu que a Microsoft derrubasse a limitação, sem com isso quebrar a compatibilidade com os softwares antigos.

NTFS

O NTFS é um sistema de arquivos mais antigo do que muitos acreditam. Ele começou a ser desenvolvido no início da década de 1990, quando o projeto do Windows NT dava os seus primeiros passos.

A idéia foi desde o início, criar um sistema de arquivos que pudesse ser usado durante décadas, por mais que os discos rígidos evoluíssem.

Já que o grande problema do sistema FAT16 era o fato de serem usados apenas 16 bits para o endereçamento de cada cluster, permitindo apenas 65 mil clusters por partição, o NTFS incorporou desde o início a capacidade para endereçar os clusters usando endereços de 64 bits. A única limitação agora passa a ser o tamanho dos setores do HD. Como cada setor possui 512 bytes, o tamanho de cada cluster usando NTFS também poderá ser de 512 bytes,

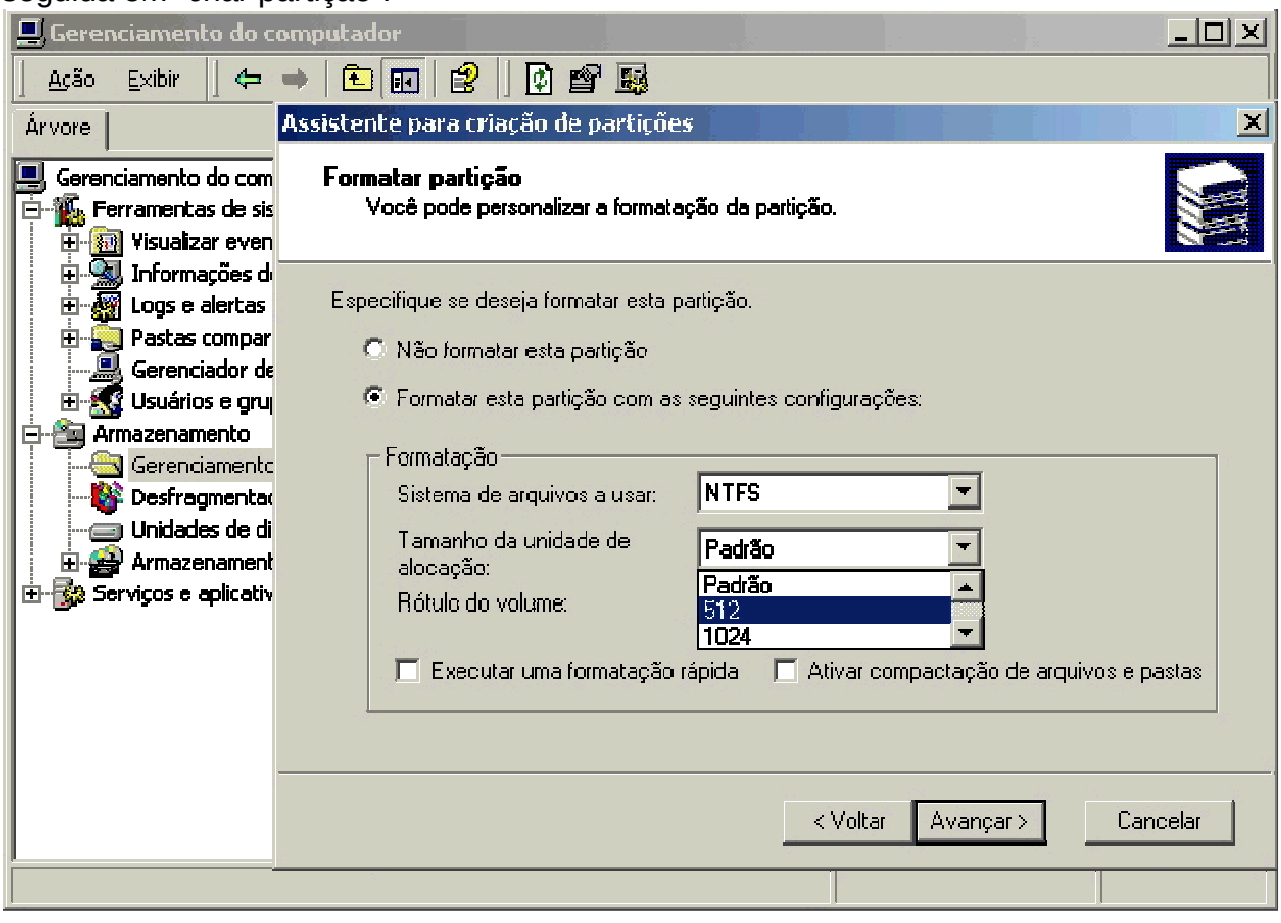
independentemente do tamanho da partição.

É sem dúvida um grande avanço sobre os clusters de 32 KB e as partições de até 2 GB da FAT 16. Mas, existe um pequeno problema em endereçar partições muito grandes usando clusters de 512 bytes: o desempenho. Com um número muito grande de clusters, o processamento necessário para encontrar os dados desejados passa a ser muito grande, diminuindo a performance.

Assim como na FAT 32, ficou estabelecido que o tamanho mínimo de clusters seria usado por default apenas em partições de um certo tamanho:

Tamanho da partição	Tamanho do cluster
até 512 MB	512 bytes
até 1 GB	1 KB
até 2 GB	2 KB
acima de 2 GB	4 KB

Apesar do default ser usar clusters de 4 KB em qualquer partição maior do que 2 GB, você pode criar partições com clusters do tamanho que desejar através do assistente para criação de partições do Windows 2000/XP, que pode ser encontrado em Painel de controle > Ferramentas Administrativas > Gerenciamento do computador > Armazenamento > Gerenciamento de disco. Do lado direito da tela será mostrado um mapa dos HDs instalados na máquina, basta clicar com o botão direito sobre uma área de espaço livre e em seguida em “criar partição”.

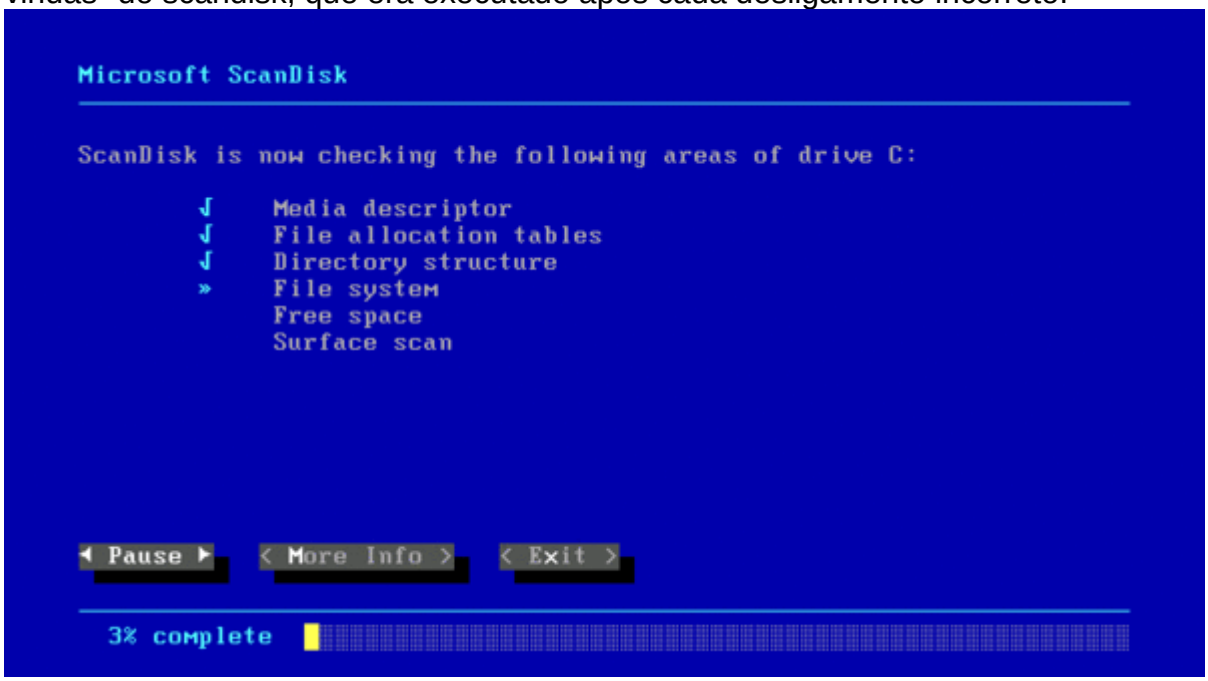


Continuando, mais uma vantagem do sistema NTFS é que os nomes de arquivos e pastas utilizam caracteres em Unicode, ao invés de ACSII. O ASCII é o sistema onde cada caracter ocupa 1 byte de dados, mas são permitidas apenas letras, números e alguns caracteres especiais. No Unicode, cada caracter ocupa dois bytes, o que permite 65 mil combinações, o suficiente para armazenar caracteres em vários idiomas. Isso permite que usuários do Japão, China, Taiwan e outros países que não utilizam o alfabeto ocidental, possam criar arquivos usando caracteres do seu próprio idioma, sem a necessidade de instalar drivers e programas adicionais que adicionem o suporte.

Outro ponto importante onde o NTFS é superior ao sistema FAT é na tolerância a falhas. No sistema FAT, sempre que o sistema trava ou é desligado enquanto estão sendo atualizados arquivos os diretórios no HD, existe uma possibilidade muito grande do sistema tornar-se inconsistente, com arquivos interligados, agrupamentos perdidos e os outros problemas. Surge então a necessidade de rodar o scandisk depois de cada desligamento incorreto.

No NTFS, o sistema mantém um log de todas as operações realizadas. Com isto, mesmo que o micro seja desligado bem no meio da atualização de um arquivo, o sistema poderá, durante o próximo boot, examinar este log e descobrir exatamente em que ponto a atualização parou, tendo a chance de automaticamente corrigir o problema. Além de reduzir a perda de tempo, a possibilidade de perda de dados é muito menor.

Se você chegou a usar o Windows 95/98/ME, deve lembrar-se da "tela de boas vindas" do scandisk, que era executado após cada desligamento incorreto:



Clusters contendo setores defeituosos também são marcados automaticamente, conforme são detectados, sem a necessidade de usar o scandisk ou qualquer outro utilitário. Neste caso, a marcação é feita na tabela de endereçamento da partição, de forma que a lista de setores defeituosos é perdida ao reparticionar o HD. Antigamente, os HDs eram menos confiáveis e o

aparecimento de setores defeituosos um fenômeno muito mais comum, de forma que a maioria dos programas de formatação realizavam um teste de superfície durante a formatação da partição (como no caso do format usado no Windows 95/98, onde formatar uma partição podia demorar mais de uma hora ;). Atualmente, a maioria dos programas realiza uma formatação rápida, presumindo que o HD não possua setores defeituosos.

Existiram diversas versões do NTFS, que acompanharam a evolução do Windows NT. A partir do Windows 2000, foi introduzido o NTFS 5, que trouxe diversos aperfeiçoamentos, incluindo o suporte ao Active Directory.

Outro recurso interessante é a possibilidade de encriptar os dados gravados, de forma a impedir que sejam acessados por pessoas não autorizadas mesmo caso o HD seja removido e instalado em outro micro. Este recurso de encriptação é interessante, por exemplo, para profissionais de campo, que levam dados secretos em seus laptops. É possível tanto criptografar o disco inteiro, quanto pastas ou arquivos individuais.

Também é possível compactar pastas e arquivos individuais, economizando espaço em disco. No Windows 95/98 era possível compactar partições usando o drvspace, mas só era possível compactar partições inteiras, o que normalmente acaba não sendo um bom negócio, pois diminuia bastante a velocidade do micro e aumentava a possibilidade de perda de dados. Naturalmente, a compactação também é diferente da feita por programas como o Winzip, já que os arquivos e pastas continuam acessíveis exatamente da mesma forma, com o sistema fazendo a compactação e descompactação do arquivo de forma transparente.

Com a possibilidade de compactar pastas individuais, você pode comprimir apenas as pastas contendo um grande volume de arquivos que suportam um bom nível de compressão, deixando de lado pastas com fotos, músicas e arquivos de vídeo, arquivos que já estão comprimidos. Para compactar uma pasta, acesse o menu de propriedades. Na seção "avançadas", marque a opção de compactar arquivos para economizar espaço.

A compactação de arquivos exige uma carga adicional de processamento, já que o sistema tem o trabalho de descompactar os arquivos antes de acessá-los. Antigamente, usar compactação reduzia muito o desempenho do sistema, já que os processadores eram mais lentos. Num micro atual, a redução é muito menos significativa e, em muitos casos, o uso da compactação pode até mesmo melhorar o desempenho, já que arquivos compactados ocupam menos espaço e, consequentemente, são lidos mais rapidamente pela cabeça de leitura.

Estruturas lógicas do NTFS

Assim como no sistema FAT, no NTFS são incluídas várias estruturas lógicas no HD. Apesar da idéia ser basicamente a mesma, estas estruturas são bem diferentes no NTFS.

Em primeiro lugar, temos a **MFT** (Master File Table), que substitui a FAT, armazenando as localizações de todos os arquivos e diretórios, incluindo os arquivos referentes ao próprio sistema de arquivos. Mas, a forma como este mapeamento é feito difere um pouco do sistema FAT.

Cada entrada de arquivo ou diretório no MFT possui 2 KB, onde são armazenados o nome do arquivo e seus atributos. Sobra então uma pequena área de dados, geralmente de 1500 bytes (pode ser maior ou menor, dependendo do espaço ocupado pelo nome e pelos atributos do arquivo) que é usada para guardar o início do arquivo.

Caso o arquivo seja muito pequeno, ele poderá ser armazenado diretamente na entrada no MFT. Caso contrário, serão armazenados apenas os números dos clusters ocupados pelo arquivo.

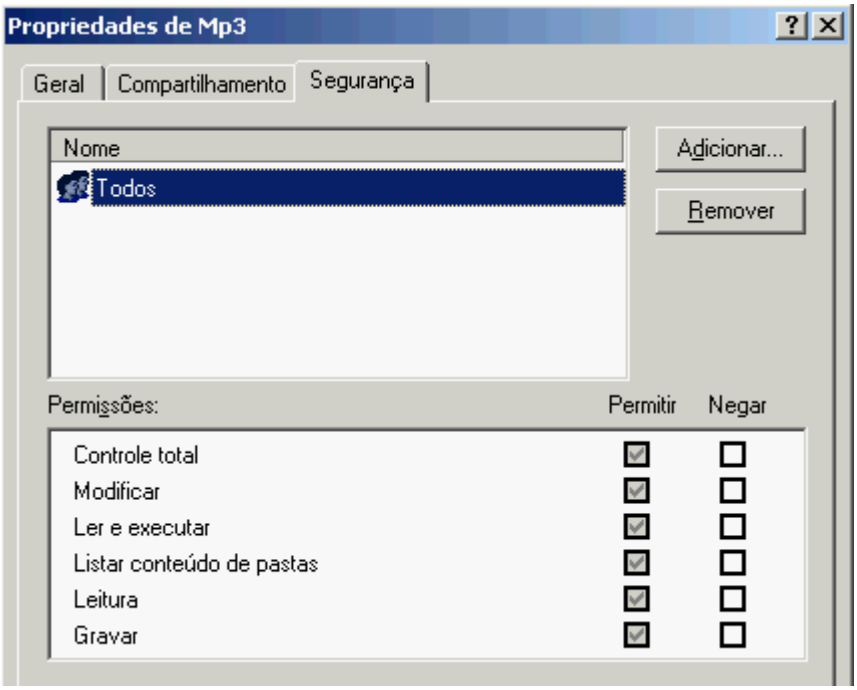
Em alguns casos, não é possível armazenar nem mesmo os atributos do arquivo no MFT, neste caso, os atributos serão gravados em clusters vagos do HD e a MFT conterá apenas entradas que apontam para eles. Pode parecer estranho que um arquivo possa ter mais de 2 KB só de atributos, mas no NTFS os atributos do arquivo vão muito além dos atributos de arquivo, diretório, oculto, etc. que existem no sistema FAT.

Os atributos do arquivo incluem seu nome, versão, nome MS-DOS (o nome simplificado com 8 caracteres e extensão), mas, principalmente incluem as permissões do arquivo, quais usuários do sistema poderão acessa-lo ou não, e ainda um espaço reservado para auditoria, que permite armazenar informações sobre quais operações envolvendo o arquivo devem ser gravadas para que seja possível realizar uma auditoria, caso necessário.

Em seguida, temos a questão das **permissões de acess**o, uma parte importante da configuração de um servidor, ou de qualquer máquina que vá ser utilizada por diversos usuários.

Para configurar as permissões de acesso, abra a guia “Segurança”. As configurações valem tanto para acesso locais, quanto acessos através da rede. O Windows aplica as permissões de acesso de acordo com o usuário logado na máquina.

Por default, todos têm acesso total à pasta. Você verá no campo “nome” o grupo “todos” e todas as permissões marcadas como “permitir”. O grupo “todos” significa todos os usuários do sistema. Se você quiser um melhor controle, pode deletar o grupo “todos” e adicionar um a um os usuários que terão acesso à pasta.



Depois de fazer isso, clique sobre o usuário para configurar as suas permissões, que aparecem na parte de baixo da janela. Você pode até mesmo configurar as permissões de modo que nem você mesmo possa acessar a pasta :). Neste caso, você receberá uma mensagem de acesso negado até voltar e reconfigurar as permissões.

A configuração das permissões pode ser a parte mais importante da implantação de uma rede baseada no Windows 2000, XP ou mesmo do antigo NT ao mesmo tempo em que pode ser de longe a mais trabalhosa, dependendo do número de usuários e restrições que tiver de configurar.

É possível também estabelecer quotas de disco, que limitam a quantidade de espaço que determinados usuários podem utilizar. Este recurso é muito utilizado em servidores web e em servidores de arquivo, mas pode ser útil também em situações mais corriqueiras, como quando você precisa limitar quanto espaço seu irmão menor pode usar, evitando que ele entupa o HD de downloads. ;)

O NTFS inclui também recursos que reduzem de forma significativa a fragmentação do sistema de arquivos. Quando existe um volume considerável de espaço em disco, o sistema reserva até 15% do espaço da partição para armazenar o MFT e as entradas referentes aos atributos dos arquivos, de forma que todas as informações possam ser gravadas em setores contínuos. Os próprios arquivos são salvos de forma inteligente, com o sistema dando preferência a áreas onde ele pode ser gravado seqüencialmente, sem fragmentação.

Apesar disso, com o passar dos meses, é normal que um certo nível de fragmentação ocorra, de forma que um desfragmentador vem a calhar. Tanto o Windows 2000 e XP, quanto o Vista incluem um desfragmentador capaz de lidar com partições NTFS. Apesar de demorado, vale à pena usá-lo de vez em quando.

Com relação ao desempenho, existe uma certa polêmica, já que por ser mais complexo, o NTFS é realmente mais lento que o sistema FAT em micros muito antigos, ou quando são manipuladas pequenas quantidades de arquivos. Este é um dos motivos dele ser utilizado apenas em micros PCs e não em câmeras e celulares, por exemplo, onde o processamento necessário seria proibitivo.

Apesar disso, o NTFS é capaz de lidar eficientemente com volumes muito maiores de arquivos, de forma que a balança tende para o outro lado em cenários mais complexos, como na bagunça de arquivos, e-mails, fotos, músicas e arquivos temporários que é um desktop atual. :)

Por exemplo, o NTFS utiliza o conceito de balanced trees (árvores balanceadas ou B+ trees), onde as informações relacionadas a cada diretório são gravadas próximas umas das outras, ao invés de ficarem numa tabela central, como no sistema FAT. A cabeça de leitura precisa percorrer uma distância maior para acessar as informações relacionadas a cada diretório, mas em compensação perde menos tempo lendo informações referentes aos arquivos dentro deste diretório, o que acaba compensando a perda inicial e até revertendo em ganho, que se torna mais e mais expressivo confirme cresce o volume de arquivos e pastas armazenados.

Concluindo, temos o LFS (Log File Service), que é o principal responsável pela tolerância à falhas do sistema NTFS. Tolerância a falhas neste caso significa não perder dados ou estruturas do sistema de arquivos quando o sistema travar, ou houver qualquer outro imprevisto, ou que, pelo menos, o estrago seja o menor possível.

Para isso, o sistema mantém um log com todas as alterações feitas no sistema de arquivo. Ao gravar um arquivo qualquer por exemplo, será primeiro gravada uma entrada no log, com os detalhes sobre a operação, qual arquivo está sendo gravado, em que parte do disco, etc. ao terminar a gravação é gravada uma outra entrada, um OK confirmando que tudo deu certo. Caso o sistema seja desligado incorretamente durante a gravação, é possível verificar no próximo boot o que estava sendo feito e fazer as correções necessárias. Periodicamente, o sistema verifica todas as entradas do Log e caso esteja tudo em ordem, deleta o antigo log, para evitar que o arquivo ocupe muito espaço em disco.

EXT3

O EXT3 é atualmente o sistema de arquivos mais utilizado no mundo Linux. Usado por padrão pela grande maioria das distribuições.

Tudo começou com o sistema EXT (Extended File System), introduzido em 1992. Nos estágios primários de desenvolvimento, o Linux utilizava um sistema de arquivos bem mais antigo, o MinixFS. O Minix é um sistema Unix, que Linux Torvalds usou como base nos estágios primários do desenvolvimento do Linux. Entretanto, o MinixFS possuía pesadas limitações, mesmo para a época. Os endereços dos blocos de dados tinham apenas 16 bits, o que permitia criar

partições de no máximo 64 MB. Além disso, o sistema não permitia nomes de arquivos com mais de 14 caracteres.

Não é de se estranhar que, em pouco tempo o Linux ganhou seu sistema de arquivos próprio, o “Extended File System”, ou simplesmente EXT, que ficou pronto em Abril de 92 a tempo de ser incluído no Kernel 0.96c.

Nesta primeira encarnação, o EXT permitia a criação de partições de até 2 GB e suportava nomes de arquivos com até 255 caracteres. Foi um grande avanço, mas o sistema ainda estava muito longe de ser perfeito. O desempenho era baixo e ele era tão sujeito a fragmentação de arquivos quanto o sistema FAT. Além disso, logo começaram a surgir HDs com mais de 2 GB, de forma que em 1993 surgiu a primeira grande atualização, na forma do EXT2.

O EXT2 trouxe suporte a partições de até 32 TB, manteve o suporte a nomes de arquivos com até 255 caracteres, além de diversos outros recursos.

O maior problema do EXT2 é que ele não inclui nenhum sistema de tolerância a falhas. Sempre que o sistema é desligado incorretamente, é necessário utilizar o fsck, um utilitário similar ao scandisk do Windows, que verifica todos os blocos do sistema de arquivos, procurando por inconsistências entre as estruturas e descrições e os dados efetivamente armazenados.

O teste do fsck demora bastante (bem mais que o scandisk) e o tempo cresce proporcionalmente de acordo com o tamanho da partição. Num HD atual, o teste pode, literalmente, demorar horas.

Este problema foi corrigido com o EXT3, que foi introduzido em 1999. A principal característica do EXT3 é o uso do recurso de journaling, onde o sistema de arquivos mantém um journal (diário) das alterações realizadas, um recurso similar ao LFS usado no NTFS.

Este “diário” armazena uma lista das alterações realizadas, permitindo que o sistema de arquivos seja reparado de forma muito rápida após o desligamento incorreto. O fsck continua sendo usado, mas agora ele joga de acordo com as novas regras, realizando o teste longo apenas quando realmente necessário.

O EXT3 possui três modos de operação:

No modo **ordered** (o default), o journal é atualizado no final de cada operação. Isto faz com que exista uma pequena perda de desempenho, já que a cabeça de leitura precisa realizar duas operações de gravação, uma no arquivo que foi alterada e outra no journal (que também é um arquivo, embora especialmente formatado) ao invés de apenas uma.

No modo **writeback** o journal armazena apenas informações referentes à estrutura do sistema de arquivos (metadata) e não em relação aos arquivos propriamente ditos, e é gravado de forma mais ocasional, aproveitando os momentos de inatividade. Este modo é o mais rápido, mas em compensação

oferece uma segurança muito menos contra perda e corrompimento de arquivos causados pelos desligamentos incorretos.

Finalmente, temos o modo **journal**, que é o mais seguro, porém mais lento. Nele, o journal armazena não apenas informações sobre as alterações, mas também uma cópia de segurança de todos os arquivos modificados, que ainda não foram gravados no disco. A cada alteração, o sistema grava uma cópia do arquivo (no journal), atualiza as informações referentes à estrutura do sistema de arquivos, grava o arquivo e atualiza novamente o journal, marcando a operação como concluída. Como disse, isso garante uma segurança muito grande contra perda de dados, mas em compensação reduz o desempenho drasticamente. Justamente por causa disso, este é o modo menos usado.

Para usar o modo writeback ou o modo journal, você deve adicionar a opção "data=writeback", ou "data=journal" nas opções referentes à partição, dentro do arquivo "/etc/fstab".

Desta forma, ao invés de usar "/dev/hda5 /mnt/hda5 ext3 defaults 0 2", por exemplo, você usaria "/dev/hda5 /mnt/hda5 ext3 data=writeback 0 2"

O EXT3 (assim como o EXT2) utiliza endereços de 32 bits e blocos (análogos aos clusters usados no sistema FAT) de até 8 KB. Tanto o tamanho máximo da partição, quanto o tamanho máximo dos arquivos são determinados pelo tamanho dos blocos, que pode ser escolhido durante a formatação:

Tamanho dos blocos	Tamanho máximo da partição	Tamanho máximo dos arquivos
1 KB	2 TB	16 GB
2 KB	8 TB	256 GB
4 KB	16 TB	2 TB
8 KB	32 TB	2 TB

Uma observação é que, em versões antigas do Kernel, o limite para o tamanho máximo de arquivos no EXT2 já foi de 2 GB e em seguida de 16 GB, mas ambas as limitações caíram a partir do Kernel 2.6, chegando à tabela atual.

Por padrão, o tamanho do bloco é determinado automaticamente, de acordo com o tamanho da partição, mas é possível forçar o valor desejado usando o parâmetro "-b" do comando mkfs.ext3 (usado para formatar as partições EXT3 no Linux), como em "mkfs.ext3 -b 2048 /dev/hda1" (cria blocos de 2 KB) ou "mkfs.ext3 -b 4096 /dev/hda1" (para blocos de 4 KB).

Assim como no caso do NTFS, usar clusters maiores resulta em mais espaço desperdiçado (sobretudo ao guardar uma grande quantidade de arquivos pequenos) mas, além do aumento no tamanho máximo dos arquivos e partições, resulta em um pequeno ganho de desempenho, já que reduz o processamento e o número de atualizações na estrutura do sistema de arquivos ao alterar os dados gravados.

Embora o limite de 32 TB para as partições EXT3 não seja um problema hoje em dia, ele tende a se tornar um obstáculo conforme os HDs crescerem em

capacidade, assim como os limites anteriores. Para evitar isso, o EXT4, legítimo sucessor do EXT3, incorporou o uso de endereços de 48 bits, o que permite endereçar um volume virtualmente ilimitado de blocos. Só para referência, o EXT4 permite criar partições de até 1024 petabytes :). O limite de 2 TB para os arquivos também foi removido, abrindo espaço para o armazenamento de bases de dados gigantes e outros tipos de arquivos que eventualmente venham a superar esta marca.

Embora existam diversos outros sistemas de arquivos para o Linux, como o ReiserFS, XFS, JFS e assim por diante, o EXT3 continua sendo o sistema de arquivos mais utilizado, já que ele atende bem à maioria e é muito bem testado e por isso bastante estável. A tendência é que o EXT3 seja lentamente substituído pelo EXT4 e os demais sistemas continuem entrincheirados em seus respectivos nichos.

Sistema de Arquivo de Rede (NFS - Network File System)

NFS (acrônimo para **Network File System**) é um sistema de arquivos distribuídos desenvolvido inicialmente pela Sun Microsystems, Inc., a fim de compartilhar arquivos e diretórios entre computadores conectados em rede, formando assim um diretório virtual. O protocolo Network File System é especificado nas seguintes RFCs: RFC 1094, RFC 1813 e RFC 3530 (que tornou obsoleta a RFC 3010).

Finalidade

O cliente NFS tem por finalidade tornar o acesso remoto transparente para o usuário do computador, e esta interface cliente e servidor, executada pelo NFS através dos protocolos Cliente-Servidor, fica bem definida quando o usuário ao chamar um arquivo/diretório no servidor, lhe parece estar acessando localmente, sendo que está trabalhando com arquivos remotos.

Para que os clientes tenham acesso aos arquivos, é feita uma requisição ao servidor que, dependendo das permissões do cliente, responde confirmando a requisição. A partir desse ponto a hierarquia de diretórios e arquivos remotos passa a fazer parte do sistema de arquivos local da máquina.

Existe neste ponto uma relação com o Sistema de Nomeação de Arquivos, pois há a necessidade de se criar o endereço daqueles arquivos ou diretórios. Este sistema cuida de identificar a localização de um determinado arquivo ou diretório, quando se é fornecido seu nome ou caminho. Para isso o sistema deve oferecer uma resolução por nomes (mapeamento de nomes de arquivos legíveis por humanos – strings, para nomes de arquivos legíveis por máquinas – números manipuláveis por máquinas) ou resolução por localização (mapeamento de nomes globais para uma determinada localização), ou ainda, ambas.

Utilização

Um exemplo da utilização do NFS é a disponibilização das áreas de trabalho dos usuários em toda a rede e, quando o este efetua o login, seu diretório de trabalho pode ser acessado via NFS. Supondo que o usuário mude de estação de trabalho, o seu diretório pode ser disponibilizado novamente nesta estação e sem que nenhuma configuração adicional seja realizada.

Sua interface é pública e muito utilizada para o compartilhamento de leituras e organizações acadêmicas, pelas vantagens de, entre outras: transparência; unificação de comandos; redução de espaço local; independência de sistemas operacionais e hardware.

Para um sistema cliente-Servidor, o cliente pode sempre que logar na máquina "importar" automaticamente os diretórios e arquivos que o mesmo criou na sua área pessoal, por exemplo (para implementar esse sistema de importacão de arquivo associado a um usuário em específico é necessário ter configurado um Sistema com LDAP ou NIS, além do NFS).

Implementação

Para que os clientes possam acessar o servidor NFS é necessário que os seguintes daemons estejam executando[1]:

nfsd - daemon NFS, que atende requisições dos clientes NFS.

mountd - daemon de montagem NFS, que executa as solicitações que o nfsd lhe passa.

portmap - daemon portmapper, permite que clientes NFS descubram qual porta o servidor NFS está utilizando.