Livros:

- Sistemas Operacionais Modernos - Tanembaun
- Silbershatz - Livro Texto
- Sibershatz - Livro de Solução de Exercícios

MEMÓRIA VIRTUAL acho que compreende as aulas 18,19,20 do moodle

Todas estratégias de gerência de memória tem o mesmo objetivo: manter muitos programas na memória simultaneamente, para aumentar o grau de multiprogramação.

A memória física pode ser organizada em quadros de página, e os quadros de página físicos atribuídos a um processo podem não ser contíguos.
Fica a critério da mmu associar as páginas lógicas aos quadros de página físicos na memória.

Paginação por demanda: tu carrega só o que é necessário, imagina um programa que a primeira tela é um menu, tu não precisa carregar tudo.
Só aquilo que o usuário for selecionar. Ou seja, carrega as páginas a medida que forem necessárias. Logo, as páginas que nunca são acessadas,
nunca são carregadas para a memória física. => NUNCA TRAZER UMA PÁGINA PARA A MEMÓRIA ATÉ SER NECESSÁRIO.

Precisamo de um jeito de saber se a página está na memória ou no disco. Para isso utilizamos um bit para informar.
Valido = página é válida e está na memória. | Inválida = Não está no espaço de endereços lógicos do processo ou é válida mas estã no disco.

Se o processo tentar acessar uma página que não foi trazida para a memória, o hardware de paginação, ao traduzir o endereço na tabela de página, notará que o bit de inválido está marcado, causando uma intercepção para o Sistema Operacional que vai trazer a página para a memória e reiniciar a instrução que foi interrompida pela interceptação.

O comportamento de que um processo chamaria várias páginas que não estão na memória a cada instrução é improvável (a situação tornaria o desempenho do sistema inaceitável). Os programas costumam ter localidade de referência (9.6.1), que resulta em um desempenho razoável em relação à paginação por demanda.

O HW necessário para a paginação por demanda é o mesmo que o HW para a paginação e swap: Tabela de páginas e Memória Secundária.
Um requisito crucial é a habilidade de reiniciar qualquer instrução após uma falha de página.

É importante manter baixa a taxa de falha de página em um sistema de paginação por demanda. Caso contrário, o tempo de acesso efetivo aumenta, atrasando bastante a execução do processo.

Copy on write, funciona permitindo que os processos pai e filho compartilhem as mesmas páginas. Essas páginas compartilhadas são marcadas como paginas de cópia na escrita, significando que, se um dos processos escrever em uma página compartilhada, uma cópia da página compartilhada será criada. Só cria uma cópia quando modificamos uma das páginas.

Superalocação, ocorre uma falha de página, o SO determina onde a página está residindo no disco, mas descobre que não existem quadros livres na lista de quadros livres: toda a memória

está sendo utilizada.

Substituição de página: se nenhum quadro estiver livre, encontraremos um que não esteja sendo usado e o liberaremos. Podemos liberar um quadro escrevendo o seu conteúdo no swap space a alterando a tabela de página (e todas as outras tabelas) para indicar que a página não está mais na memória. Agora podemos usar o espaço liberado para mantera a página da qual o processo notou a falha.
A fim de melhorar o desempenho (tendo em vista que agora a falha implica em 2 transferencias de página, uma in e outra out) temos um bit de modificação que indica se a página em questão foi modificada desde que veio do disco. Se sim, precisamos gravar a "versão nova" no disco, se não não precisamos fazer nada, pois ela já está no disco, melhorando assim o desempenho.

Com a paginação por demanda, o tamanho do espaço de endereços lógico não é mais restrito pela memória física.

Algoritmos de Substituição: Podem ser globais ou locais
- **Globais**: Pode substituir páginas de qualquer processo.
- **Locais**: Só substitui páginas do processo que teve page-fault.

Algoritmo de substituição FIFO: associa a cada página a hora em que essa página foi trazida para a memória. Quando a página tiver de ser substituída, a página mais antiga será a escolhida. Podemos fazer isso implementando uma fila, substituindo sempre a que estiver na primeira posição e acrescentando ao final da fila. Seu desempenho nem sempre é bom. Uma escolha de substituição ruim aumenta a taxa de falhas de página e atrasa a execução do processo, no entanto, não causa execução incorreta.

anomalia de Belady: para alguns algoritmos de substituição de página, a taxa de falha de página pode aumentar enquanto a quantidade de quadros alocados aumenta.

Algoritmo de substituição de página ótima: possui a menor taxa de falha de página de todos os algoritmos. Sua idéia é simples: substitua a página que não será usada pelo maior período. Difícil de implementar, pois exige conhecimento futuro da string de referência. Porém é usado como referência e benchmarking

Algoritmo de substituição LRU (Usado no Android): substituir a página que não foi usada pelo maior período (Least Recently Used). Associa a cada página a hora do último uso da página, quando precisar substituir, escolhe a página que não foi usada pelo maior período. Não tem como implementar LRU sem a assistência do HW além dos registradores de TLB-padrão.
Poucos sistemas proveem suporte de HW suficiente para verdadeira substituição de página LRU. Entretanto muitos sistemas proveem alguma ajuda na forma de um bit de referência. Inicialmente todos os bits são zerados, a medida que um processo do usuário é executado, o bit associado a cada página referenciada é marcado com 1 pelo HW. Depois, podemos determinar quais páginas foram usadas e quais não foram, examinando os bits de referência, embora não saibamos a ordem do uso. Essa informação é a base para muitos algoritmos de substituição de página, que se aproximam da substituição LRU.

Algoritmo de bits de referência adicionais: mantemos um byte para cada página em uma

dada tabela de memória, em intervalos regulares uma interrupção do temporizador transfere o controle para o SisOp, o SisOp desloca o bit de referência de cada página para o bit de alta ordem do seu byte, deslocando os outros bits para a direita em 1 bit e descartando o bit de baixa ordem. Esses registradores conterão o histórico do uso a página. Ex: se o registrador de deslocamento estiver 0000 0000, então a página não foi usada por oito períodos de tempo. uma página usada pelo menos uma vez a cada período: 1111 1111. Uma página com valor de registrador de histórico igual a 1100 0100 foi usada mais recentemente do que uma com um valor 0111 0111. A quantidade de bits do histórico pode variar.

Algoritmo da Segunda Chance: é um algoritmo de substituição FIFO, entretanto, quando uma página tiver sido selecionada, inspecionamos seu bit de referẽncia. Se o valor for 0, prosseguimos substituindo essa página, mas, se o bit de referência for definido como 1, damos uma segunda chance à página e selecionamos a próxima página FIFO. Quando uma página recebe uma segunda chance, seu bit de referência é apagado e sua hora de chegada é reiniciada para a hora de chegada atual, assim uma página que recebe uma segunda chance não será substituída até todas as outras páginas terem sido substituídas (ou recebido uma segunda chance). Se uma página for usada com frequencia suficiente para manter seu bit de referência marcado, ela nunca será substituída. Um modo de implementar o algoritmo é com uma fila circular. No pior dos casos ele vai percorrer toda a fila e voltar para o primeiro caso, que recebeu uma segunda chance, agora com o bit zerado, sendo substituído.

Algoritmo da Segunda Chance melhorado: podemos melhorar o algoritmo da segunda chance considerando o bit de referência e o bit de modificação como um par ordenado, com esses dois bits, temos as quatro classes possíveis a seguir:

0 0 -> nem usada recentemente, nem modificada => melhor pagina para substituir

0 1 -> não usada recentemente, mas modificada => não tão bom, pois página precisará ser escrita antes de substituir

1 0 -> usada recentemente, mas limpa => é provável que logo seja usada novamente

1 1 -> usada recentemente e modificada => provavel que seja usada novamente, tem que escrever no disco antes de substituir.

Substituímos a primeira página encontrada na classe não-vazia mais baixa (de 00 para 11). Logo podemos ter de varrer uma fila circular várias vezes antes de encontrar uma página a ser substituída. A principal diferença para a versão mais simples é que aqui, damos preferência às páginas **não** modificadas, para reduzir a quantidade de E/S necessária.

Algoritmos de alocação: a forma mais simples de dividir m quadros entre n processos é dar a cada um uma fatia igual, m/n quadros, esse esquema é chamado de alocação igual (e**qual allocation**).
Uma alternativa é reconhecer que diversos processos precisarão de diferentes quantidades de memória. Para resolver esse problema podemos utilizar a alocação proporcional (**proportional allocation**), em que alocamos a memória disponível a cada processo de acordo com seu tamanho. Dessa maneira os processos compartilham os quadros disponíveis de acordo com suas necessidades e não igualmente.

Com vários processos competindo pelos quadros, podemos classificar os algoritmos de substituição de páginas em duas grandes categorias:

***SUBSTITUIÇÃO GLOBAL:** O algoritmo pode selecionar páginas para page-out de qualquer processo, mesmo que um determinado processo nunca tenha tido page-fault. “*Um processo*

pode 'pagar' pelo page-fault de outro"

***SUBSTITUIÇÃO LOCAL:** O algoritmo pode selecionar páginas para page-out somente a partir do conjunto de quadros alocados do processo que teve page-fault.

Na substituição global, um processo com maior prioridade pode selecionar quadros de processos de baixa prioridade para substituição, assim permitindo que um processo de alta prioridade aumente sua alocação de quadros à custa de um processo de prioridade baixa. Um problema com o algoritmo de substituição global é que um processo não pode controlar sua própria taxa de falhas (porque um outro pode roubar os seus quadros enquanto ele está de bobeira).
A substituição global atrapalha um processo, não tornando disponível para ele outras páginas na memória.
Sendo assim, a substituição global em geral resulta em maior throughput e é o método mais usado.

Thrashing: uma alta frequência de page-faults em que um processo está gastando mais tempo paginando (está sempre tendo que trocar, e depois trazer de volta) do que executando, resultando em graves problemas de desempenho, a utilização da CPU cai bruscamente. Nesse ponto, para aumentar a utilização da CPU e acabar com o thrashing, temos de diminuir o grau de multiprogramação. Podemos limitar os efeitos do thrashing usando um algoritmo de substituição local, pois um processor não poderá roubar de outro processo e fazer com que o último também cause trashing. Entretanto o problema não está solucionado.
Para impedir o thrashing, temos de fornecer a um processo tantos quadros quantos ele precisa. Para sabermos quantos ele precisa, começamos examinando quantos quadros um processo está usando. Essa técnica define o modelo de localidade da execução do processo. O modelo de localidade afirma que, à medida que um processo é executado, ele passa de uma localidade para outra. Uma localidade é um conjunto de páginas que são usadas ativamente juntas. Um programa em geral é composto de várias localidades, que podem se sobrepor. por exemplo, quando uma subrotina é chamada, ela define uma nova localidade.

O **Modelo de Conjunto de Trabalho (Working Set Model)** é baseado na suposição de localidade. Esse modelo utiliza um parãmetro delta para definir a janela de conjunto de trabalho (**Working Set Window**). A idéia é examinar as referências da página delta mais recentes. O conjunto de páginas nas referências de página delta mais recente é o conjunto de trabalho (working set). Se uma página estiver em uso ativo, ela estará no working set. Se não estiver mais sendo usada, ela sairá do working set delta unidades de tempo após sua última referência. Assim, o working set é uma aproximação da localidade do programa. A precisão do working set depende de delta, se for muito pequeno, ele não abrangerá a localidade inteira; se for muito grande, ele poderá sobrepor várias localidades. Se delta for infinito o working set será o conjunto de todas as páginas referenciadas durante a execução do processo.
A propriedade mais importante do working set é o seu tamanho.
Quando delta tiver sido selecionado o uso do modelo de conjunto de trabalho será simples: o sisop monitora o conjunto de trabalho de cada processo e aloca a esse conjunto de trabalho quadros suficientes para fornecer o tamanho do seu conjunto de trabalho. se houver quadros extras suficientes, outro processo poderá ser iniciado. Se a soma dos tamanhos do working set aumentar, ultrapassando a quantidade de quadros disponíveis, o sisop selecionará um

processo para suspender.
Essa estratégia de WSM impede o thrashing e mantem o grau de multiprogramação o mais alto possível, otimizando o uso da CPU.

Frequencia de falha de página (Page Fault Frequency - PFF), um modo de controlar (não desajeitado, referindo-se ao WSM) o thrashing com um caminho mais direto. Queremos controlar a taxa de falha de página. Quando for muito alta, sabemos que o processo precisa de mais quadros. Se for muito baixa, o processo pode ter muitos quadros. Podemos estabelecer limites superior e inferior para a taxa de falha de página desejada. Se a taxa de falha ultrapassar o limite máximo, alocamos outro quadro para esse processo; se ficar abaixo do limite mínimo, removemos um quadro. Assim podemos prevenir a taxa de falha de página para prevenir o thrashing. Se a taxa de falha aumentar e não tivermos quadro livres, temos de selecionar algum processo e suspendê-lo. Os quadros liberados são distribuídos aos processos com altas razões de falha de página.

Sistema buddy, vai dividindo por 2 a memória até que tenhamos um segmento que caiba o que queremos alocar e que ele não seja MENOR que o segmento. Por exemplo, para um segmento de 33kb, precisamos de um segmento de 64kb para alocar.

Uma propriedade óbvia da paginação por demanda pura é a grande quantidade de faltas de páginas que ocorre quando um processo é iniciado. Essa situação resulta na tentativa de levar a localidade inicial para a memória. A pré-paginação é uma tentativa de evitar esse alto nível de paginação inicial. A estratégia é trazer para a memória de uma só vez todas as páginas necessárias.
Em um sistema usando o modelo de WSM, mantemos com cada processo uma lista das páginas no seu conjunto de trabalho. Se tivermos de suspender um processo, llembramos o conjunto de trabalho para esse processo. Quando o processo tiver de ser retomado, automaticamente trazemos de volta para a memória seu conjunto de trabalho inteiro, antes de reiniciar o processo.
A pré-paginação pode prover uma vantagem em alguns casos, a questão é se o custo do uso da pré-paginação é menor do que o custo de atender às falhas de pagina correspondentes. Pode muito bem acontecer de muitas das páginas trazidas para a memória pela pre-paginação não serem utilizadas.

Interlock de E/S, protege quadros crucias contra a susbstituição global. Um bit de lock está associado a cada quadro. Quando uma página é selecionada para substituição, seu bit lock é ligado; ele permanece assim até o processo de falha de página ser novamente despachado.
O uso do bit lock pode ser perigoso, se ele nunca for desligado o quadro bloqueado se tornará inutilizável.

**** ENTRADA E SAÍDA **** compreende a aula 21

O papel do Sisop em relação ao I/O do computador é gerenciar e controlar as operações e os dispositivos de I/O.
Um dispositivo comunica-se com um computador enviando sinais através de um cabo (ou mesmo o ar), e comunica-se com a máquina por intermédio de um ponto de conexão (ou porta) como, por exemplo, uma porta serial. Se um ou mais dispositivos utilizarem um conjunto comum de fios, a conexão é chamada de BUS.
BUS é um conjunto de fios e um protocolo rigidamente definido que especifica um conjunto de mensagens que podem ser enviadas pelos fios.
Quando um dispositivo A possui um cabo que se liga ao dispositivo B que se liga a um dispositivo C que se liga à porta do computador, este esquema é chamado de CADEIA MARGARIDA, que normalmente opera como um BUS.
Os BUSES são amplamente utilizados na arquitetura dos computadores.
Um CONTROLADOR é uma coleção de dispositivos eletrônicos que podem operar uma porta, um bus ou um dispositivo. Alguns dispositivos possuem seus próprios controladores embutidos (como um controlador de disco).
Uma porta de I/O compõe-se tipicamente de quatro registradores chamados registradores de ESTADO, de CONTROLE, de DADOS DE ENTRADA e de DADOS DE SAÍDA.
*ESTADO: contém bits que podem ser lidos pelo hospedeiro. Indicam se o comando em curso foi concluído, se um byte está disponível para ser lido a partir do registrador de dados de entrade e se houve um erro de dispositivo.
*CONTROLE: pode ser gravado pelo hospedeiro para dar início a um comando ou para alterar a modalidade de um dispositivo.
*DADOS DE ENTRADA: é lido pelo hospedeiro para obter a entrada.
*DADOS DE SAÍDA: é gravado pelo hospedeiro para enviar a saída.

Busy-waiting ou polling: quando o host está em loop, lendo o registrador de status o tempo todo até o bit ocupado ser desligado.
Para alguns dispositivos o host precisar atender rapidamente ou então dados serão perdidos. Por exemplo, quando dados estão fluindo em uma porta serial ou de um teclado, o pequeno buffer no controlador estoura e os dados são perdidos se o host esperar muito tempo antes de voltar a ler os bytes.
Em muitas arquiteturas, três ciclos de instrução de CPU são suficientes para consultar um dispositivo: ler um registrador do dispositivo, realizar o AND lógico para extrair um bit de status e desviar se não for zero. Nitidamente a operação básica de polling é eficiente, contudo se torna ineficiente quando é tentado de forma repetida. Nesses casos pode ser mais eficiente arrumar as coisas para o controlador de HW notificar a CPU quando o dispositivo estiver pronto para ser atendido, em vez de exigir que a CPU consulte sequencialmente o término de E/S. O mecanismo de HW que permite a um dispositivo notificar a CPU é chamado de INTERRUPÇÃO.

INTERRUPÇÃO funciona da seguinte maneira: o HW da CPU possui um fio chamado LINHA DE REQUISIÇÃO DE INTERRUPÇÃO (interrupt request), que a CPU percebe depois de executar cada instrução. Quando a CPU detecta que um controlador enviou um sinal na linha de requisição de interrupção, ela realiza um salvamento de estado e desvia para a rotina do tratador de interrupção, em um endereço fixo na memória. O tratador de interrupção determina a causa da interrupção, efetua a restauração do estado e executa uma instrução de retorno da interrupção para cada CPU voltar ao estado de execução anterior à interrupção. Esse

mecanismo permite que a CPU responda a um evento assíncrono, como um controlador de dispositivo ficando pronto para o atendimento.

A maioria das CPUs possui duas linhas de requisição de interrupção, uma é a INTERRUPÇÃO NÃO-MASCARÁVEL (nonmaskable interrupt), reservada para eventos irrecuperáveis como erros de memória. A segunda linha de interrupção é MASCARÁVEL: ela pode ser desativada pela CPU antes da execução de sequências de instrução críticas, que não poderão ser interrompidas.

Vetor de interrupções: contém os endereços de memória de tratadores de interrupção especializados. Tem como finalidade reduzir a necessidade de um único tratador de interrupção pesquisar todas as fontes de interrupção possíveis para determinar qual precisa ser atendida.

Encadeamento de interrupções: cada elemento no vetor de interrupções aponta para o início de uma lista de tratadores de interrupção. Quando uma interrupção é levantada, os tratadores na lista correspondente são chamados um por um, até ser encontrado um que possa atender à requisição.

O mecanismo de interrupção também implementa um sistema de níveis de prioridade de interrupção, que permite que a CPU adie o tratamento de interrupções de baixa prioridade sem abandonar todas as interrupções e torna possível uma interrupção de alta prioridade tomar o lugar da execução de uma interrupção de baixa prioridade.

O mecanismo de interrupção também é usado para tratar de uma grande variedade de exceções, como divisão por zero, acesso a um endereço de memória protegido.
Muitos sistemas operacionais utilizam o mecanismo de interrupção para paginação da memória virtual, uma falha de página é uma exceção que levanta uma interrupção, a interrupção suspende o processo atual e desvia para o tratador de falha de página no kernel do sistema. Esse tratador salva o estado do processo, move o processo para a fila de espera, realiza o gerenciamento de cache de página. escalona uma operação de E/S para buscar a página, escalona outro processo para retomar a execução e depois retorna da interrupção.

As interrupções são utilizadas em SisOps modernos para tratar dos eventos assíncronos e desviar para rotinas, em modo supervisor, no kernel. Para permitir que o trabalho mais urgente seja feito primeiro, os computadores modernos utilizam um sistema de prioridades de interrupção. Controladores de dispositivos, falhas do HW e chamadas de sistema levantam interrupções para disparar rotinas do kernel. Como as interrupções são muito usadas para processamento sensível ao tempo, é necessário o tratamento eficiente das interrupções para o bom desempenho no sistema.

A fim de evitar que o processador de uso geral fique fazendo PIO (Programmed I/O, observar bit de status e inserir dados em um registrador de controlador 1 byte de cada vez), passam parte desse trabalho para um processador específico, chamado CONTROLADOR DE ACESSO DIRETO Á MEMÓRIA (Direct Memory Access - DMA). PAra iniciar a transferência de DMA, o host escreve um bloco de comando de DMA na memória. Esse bloco contém um ponteiro para a origem de uma transferência, um ponteiro para o destino da transferência e um contador do número de bytes a serem transferidos. A CPU escreve o endereço desse bloco de comando no controlador de DMA, depois prossegue operando diretamente sobre o barramento da memória,

colocando endereços no barramento para realiar as transferências sema a ajuda da CPU principal.

Algumas memórias utilizam o ACESSO DIRETO Á MEMÓRIA VIRTUAL (Direct Virtual Memory Access - DVMA), usando endereços virtuais que passam pelo processo de tradução para endereços físicos. O DVMA pode realizar transferência entre dois dispositivos mapeados na memória sem a intervenção da CPU ou do uso da memória principal.

A finalidade da camada de driver de dispositivo é ocultar as diferenças entre os controladores de dispositivos do susbsistema de E/S do kernel. Tornar o subsistema de E/S independente do HW simplifica a tarefa do desenvolvedor de sistema operacional, e beneficia os fabricantes de HW.

Dispositivo de bloco: a interface de dispositivo de bloco captura todos os aspectos necessários para acessar unidades de disco e outros dispositivos orientados a bloco.

Socket: pense numa tomada de parede para a eletricidade: qualquer aparelho elétrico pode ser conectado. Por analogia, as chamadas de sistema na interface de socket permitem a uma aplicação criar um socket para conectar um socket local a um endereço remoto (que conecta a um socket criado por outra aplicação), para executar qualquer aplicação remota, para conectar ao socket local e para enviar e receber pacotes pela conexão. Para dar implementação dos servitores a interface de socket também provê um controle de conjunto de sockets.

Relógios e temporizadores: a maioria dos computadores possui, para prover três funções básicas: dar a hora atual, dar o tempo transcorrido, definir um temporizador para disparar a operação X no momento T.

E/S bloqueante e E/S não-bloqueante: quando uma aplicação emite uma chamada de sistema BLOQUEANTE, a execução da aplicação é suspensa, a aplicação é movida para a fila de espera, depois de a chamada de sistema ser terminada, a aplicação é movida de volta para a fila de pronto, recebendo os valores retornados pela chamada de sistema.
Alguns processos no nível de usuário precisam de E/S não-bloqueante, como programas que recebem entrade de teclado e mouse enquanto processa e exibe dados na tela. Uma chamada não-bloqueante não interrompe a execução da aplicação por um tempo estendido, em vez disso, ela retorna rapidamente, com um valor de retorno que indica quantos bytes foram transferidos.

Escalonar um conjunto de requisições de E/S significa determinar uma boa ordem em que serão executadas. A ordem em que as aplicações emitem chamadas de sistema raramente é a melhor opção. Quando uma aplicação emite uma chamada de sistema de E/S bloqueante, a requisição é colocada na fila para esse dispositivo, o escalonador de E/S arruma a ordem da fila para melhorar a eficiência geral do sistema e o tempo de resposta, além de tentar ser justo.

Quando um kernel admite E/S assíncrona, ele precisa ser capaz de registrar muitas requisições de E/S ao mesmo tempo, para isso, os SisOp pode conectar a fila de entrada a uma TABELA DE STATUS DE DISPOSITIVO, que contém uma entrada para cada dispositivo de E/S, e onde

cada entrada da tabela indica o tipo do dispositivo, endereço e estado (não funcionando, ocioso e ocupado). Se o dispositivo estiver ocupado com uma requisição, o tipo da requisição e outros parâmetros serão armazenados na entrada da tabela para esse dispositivo.

Um BUFFER é uma área da memória que armazena dados enquanto são transferidos entre dois dispositivos ou entre um dispositivo e uma aplicação. Os buffers são usados porque:
lidar com a divergência de velocidade entre o produtor e o consumidor de um fluxo de dados.
Adaptaçãpo entre dispositivos que possuem tamanhos de transferência de dados diferentes.
Admitir a semântica de cópia (quando tu faz a chamada dee escrita, por exemplo, ele cria a cópia do que tem no buffer no momento, a fim de evitar que tu aumente o espaço de bytes entre a chamada e a resposta) para a E/S da aplicação.

A diferença entre um buffer e uma cache é que um buffer pode manter a única cópia existente de um item de dados, enquanto a cache, por definição, simplesmente mantém uma cópia, em armazenamento mais rápido, de um item que reside em outro lugar.

SPOOL: é um buffer que mantém a saída para um dispositivo, como uma impressora, que não pode aceitar fluxos de dados intercalados. Ele vai enviando um por vez, quando uma aplicação termina de imprimir. Em alguns sisops o spooling é controlado por um processo daemon do sistema, em outros por uma thread do kernel. O spooling é a única maneira pela qual os sistemas operacionais podem coordenar a saída concorrente.

Via de regra, uma chamada de sistema para E/S retornará 1 bit de informação sobre o status da chamada, sinalizando o sucesso ou a falha.

Para evitar que os usuários realizem E/S ilegal, definimos todas as instruções de E/S como instruções privilegiadas, assim, os usuários não podem emitir instruções de E/S diretamente; eles precisam fazer isso com o Sisop. O sisop, executando no modo monitor, verifica e certifica-se de que a requisição é válida e, se for, realiza a operação de E/S requisitada. O sisop, em seguida, retorna ao usuário.

O UNIX tem uma tabela de montagem que associa prefixos de nomes de caminhos a nomes de dispositivo específico. Para resolver um nome de caminho, o UNIX pesquisa o nome na tabela de montagem para encontrar o maior prefixo que combina; a entrada correspondente na tabela de montagem provê o nome do dispositivo.

SISTEMA DE ARQUIVOS compreende a aula 22

O sistema de arquivos consistem em duas partes: **uma coleção de arquivos**, cada um armazenando dados relacionados, e **uma estrutura de diretório**, que organiza e fornece informações sobre todos os arquivos no sistema.

Um arquivo é uma coleção nomeada de informações relacionadas, registradas no armazenamento secundário (disco). É uma sequencia de bits, bytes, linhas ou registros, sendo que seu significado é definido pelo criador e usuário do arquivo. O conceito de arquivo é bastante geral.

Os arquivos possuem tipicamente esses atributos:

*Nome: nome do arquivo, única informação armazenada no formato legível para o ser humano.

*Identificador: normalmente um número, identifica o arquivo dentro do sistema de arquivos; ele é o nome do arquivo que não é legível para o ser humano.

*Tipo: informação necessária para sistemas que admitem diferentes tipos de arquivos.

*Local: um ponteiro para um dispositivo e para o local do arquivo nesse dispositivo.

*Tamanho: o tamanho atual do arquivo (em bytes, palavras ou blocos) e possivelmente o tamanho máximo permitido incluídos nesse atributo.

*Proteção: informações de controle de acesso determinam quem pode realizar leitura, escrita, execução e assim por diante.

*Hora, data e identificação do usuário: podem ser mantidas para a criação, última modificação e ultimo uso. Esses dados podem ser úteis para proteção, segurança e monitoramento de uso.

As informações sobre todos os arquivos são mantidas na estrutura de diretório, que também reside no armazenamento secundário. Em geral, uma entrada de diretório consiste no nome do arquivo e seu identificador exclusivo. O identificador, por sua vez, localiza os outros atributos do arquivo.

Operações de arquivo:

*Criar arquivo: é preciso encontrar um espaço no sistema de arquivos para esse arquivo, após uma aentrada para o novo arquivo precisa ser feita no diretório.

*Escrever em um arquivo: para escrever um arquivo, podemos fazer uma chamada de sistema especificando o nome do arquivo e as informações a serem escritas no arquivo. Dado o nome do arquivo, o sistema pesquisa o diretório para encontrar o local do arquivo. O sistema precisa manter um ponteiro de escrita para o local no arquivo onde acontecerá a próxima escrita. O ponteiro de escrita precisa ser atualizado sempre que houver uma escrita.

*Ler de um arquivo: para ler de um arquivo, usamos uma chamada de sistema que especifica o nome do arquivo e onde (na memória) o próximo bloco do arquivo deve ser colocado. Novamente, o diretório é pesquisado em busca da entrada associada, e o sistema precisa manter um ponteiro de leitura para o local no arquivo onde a próxima leitura deverá ocorrer. Quando a leitura tiver terminado, o ponteiro de leitura será atualizado. Como um processo ou está lendo ou está escrevendo em um arquivo, o local da operação atual pode ser mantido como um ponteiro de posição atual do arquivo por processo. As operações de leitura e escrita utilizam esse mesmo ponteiro, economizando espaço e diminuindo a complexidade do sistema.

*Reposicionar dentro de um arquivo: o diretório é pesquisado em busca da entrada correta, e o ponteiro da posição atual do arquivo é reposicionado para determinado valor. O reposicionamento dentro de um arquivo não precisa envolver qualquer E/S real. Essa operação

de arquivo também é conhecida como busca de arquivo.
*Excluir um arquivo: para excluir um arquivo, procuramos o arquivo nomeado no diretório. Ao encontrar a entrada de diretório associada, liberamos todo o espaço do arquivo, para poder ser reutilizado por outros arquivos, e apagamos a entrada do diretório.
*Truncar um arquivo: o usuário pode querer apagar o conteúdo de um arquivo, mas manter seus atributos. Em vez de forçar o usuário a excluir o arquivo e depois recriá-lo, essa função permite a todos os atributos permanecerem inalterados - exceto pelo tamanho do arquivo -, mas permite ao arquivo ser reiniciado para o tamanho zero e seu espaço ser liberado.

Os locks de arquivo permitem que um processo restrinja o acesso a um arquivo e impeça que outros processos tenham acesso a ele. São uteis para arquivos compartilhados por vários processos.

O sistema operacional mantém uma tabela de arquivos abertos, contendo informação sobre todos os arquivos abertos. Quando uma operação de arquivo é requisitada, o arquivo é especificado por meio de um índice para essa tabela, de modo que nenhuma pesquisa é necessária. Quando o arquivo não está mais sendo usado, ele é fechado pelo processo, e o sistema operacional remove sua entrada da tabela de arquivos abertos. Alguns sistemas abrem implicitamente um arquivo quando é feita a primeira referência para ele. Da mesma forma ele é fechado automaticamente quando a tarefa ou programa que abriu o arquivo termina.

Em geral, o sisop utiliza dois níveis de tabelas internas: uma tabela por processo (que acompanha todos os arquivos abertos por um processo) e uma tabela para todo o sistema. Cada entrada na tabela por processo, aponta para uma tabela de arquivos abertos para todo o sistema. A tabela para todo o sistema contém informações independentes do processo, como local do arquivo no disco, datas de acesso e tamanho de arquivo. Normalmente a tabela de arquivos abertos também possui um contador de abertura associado a cada arquivo, para indicar quantos processos abriram o arquivo. cada close() diminui esse contador, e quando chegar a zero o arquivo não está mais em uso e a entrada do arquivo será removida da tabela de arquivos abertos.

O Sisop utiliza a extensão para indicar o tipo do arquivo e o tipo das operações que podem ser feitas sobre esse arquivo.

O sistema UNIX usa um NUMERO MAGICO primitivo, armazenado no início de alguns arquivos, para indicar aproximadamente o tipo do arquivo. Nem todos possuem número mágico, de modo que os recursos so sistema não podem ser baseados unicamente nesse tipo de informação.

Como o espaço em disco é sempre alocado em blocos, alguma parte do último bloco de cada arquivo em geral é desperdiçada. Se cada bloco tivesse 512 bytes, por exemplo, então um arquivo com 1.949 bytes receberia quatro blocos (2.048 bytes); os últimos 99 bytes seriam desperdiçados. O desperdício que acontece por mantermos tudo em unidades de blocos (em vez de bytes) é a fragmentação interna. Todos os sistemas de arquivo sofrem com a fragmentação interna; quando maior o tamanho do bloco, maior a fragmentação interna.

Um lock compartilhado é quando vários processos podem obter o lock ao mesmo tempo.
Um lock exclusivo é quando apenas um processo por vez pode obter esse lock.
Sistemas operacionais podem fornecer mecanismos de lock de arquivo OBRIGATORIO ou CONSULTIVO. Se um lock for obrigatório, então, qundo um processo obtiver um lock exclusivo,

o sistema operacional impedirá que qualquer outro processo acesse o arquivo com o lock. Se o lock for de consulta, fica a critério dos desenvolvedores do software garantir que os locks sejam obtidos e liberados de forma apropriada.
O uso de lock de arquivo exige as mesmas precauções do sincronismo de processo comum.

O método de acesso a arquivo mais simples é o ACESSO SEQUENCIAL. As informações no arquivo são processadas em ordem, um registro após o outro. Esse modo é de longe o mais comum. As leituras e escritas compõe a maior parte das operações sobre um arquivo. Uma operação de leitura lê a próxima parte do arquivo e avança um ponteiro de arquivo, que acompanha o local da E/S. A escrita funciona de maneira similar.

Outro método é o ACESSO DIRETO que é baseado em um modelo de disco de um arquivo, pois os discos permitem o acesso aleatório a qualquer bloco do arquivo. Para o acesso direto, o arquivo é visto como uma sequência numerada de blocos ou registros. assim podemos ler o bloco 14, depois ler o bloco 53 e depois escrever no 7. Não existem restrições sobre a aordem de leitura ou escrita para um arquivo de acesso direto. São muito utilizados para o acesso imediato a grandes quantidades de informação.
O número de bloco fornecido pelo usuário ao sistema operacional normalmente é um NÚMERO DE BLOCO RELATIVO (um índice relativo ao início do arquivo). ASsim o primeiro bloco do aqruivo é 0, o próximo é 1,... mesmo que o endereço de disco absoluto real do bloco possa ser 92831092 para o primeiro e 20983192 para o segundo. O uso de número de blocos relativos permite que o sistema operacional decida onde o arquivo deve ser colocado e ajuda a evitar que o usuário acesse partes do sistema de arquivos que podem não fazer parte do seu arquivo.

ESTRUTURA DE DIRETÓRIO

Um sistema de arquivos pode ser criado em cada uma das partições de um disco. As partições podem ser combinadas para formar estruturas maiores, conhecidas como volumes, e os sistemas de arquivos também podem ser criados em cima destes. Cada volume pode ser considerado um disco virtual.
Cada volume que contém um sistema de arquivo também deve conter informações sobre os arquivos dentro dele. Essas informações são mantidas em entradas no DIRETÓRIO DO DISPOSITIVO ou SUMÁRIO DO VOLUME. O diretório registra informaçõeso para todos os arquivos nesse volume.

Diretório pode ser visto como uma tabela de símbolos que traduz nomes de arquivo nas entradas de diretórios.
Operações que devem ser realizadas sobre um diretório:
*Procurar um arquivo.
*Criar um arquivo.
*Excluir um arquivo.
*Listar um diretório.
*Renomear um arquivo (a troca do nome de um arquivo também pode permitir a mudança da sua posição dentro da estrutura do diretório).
*Atravessar o sistema de arquivos: podemos querer acessar cada diretório e cada arquivo dentro de uma estrutura de diretórios.

A estrutura de diretórios mais simples é a estrutura de ÚNICO NIVEL, onde todos os arquivos estão contidos no mesmo diretório. Como todos os arquivos estão no mesmo diretório eles precisam ter nomes exclusivos.

Diretório de dois níveis: na estruturua de diretório de dois níveis, cada usuário possui seu próprio DIRETÓRIO DE ARQUIVOS DO USUÁRIO (User File Directory - UFD). Os UFDs possuem estruturas semelhantes, mas cada um lista apenas os arquivos de um único usuário. Quando a tarefa de um usuário é iniciada ou quando um usuário efetua o login, o DIRETÓRIO DE ARQUIVOS MESTRE (Master File Directory - MFD) é pesquisado. O MFD é indexado por nome de usuário ou número da conta, e cada entrada aponta para o UFD desse usuário. QUando um usuário se refere a determinado arquivo, somente seu próprio UFD é pesquisado. ASsim, diferentes usuários podem ter arquivos com o mesmo nome, desde que todos os nomes de arquivo dentro de cada UFD sejam exclusivos. Para criar um arquivo para um usuário, o sistema operacional pesquisa apenas o UFD desse usuário para verificar se existe outro arquivo com esse nome.
Essa estrutura isola um usuário do outro, o isolamento é uma vantagem quando os usuários são independentes, mas é uma desvantagem quando os usuários querem cooperar em alguma tarefa e acessar os arquivos uns dos outros.

Diretórios estruturados em árvores: essa generalização permite aos usuários criarem seus próprios subdiretórios e organizarem seus arquivos de modo conveniente. A árvore possui um diretório raiz e cada arquivo no sistema possui um nome de caminho exclusivo. Um diretório contem um conjunto de arquivos e ou subdiretórios. Um bit em cada entrada de diretório define a entrada como um subdiretório (1) ou como um arquivo (0).
Os nomes de caminho podem ser de dois tipos: ABSOLUTOS e RELATIVOS. Um nome da caminho ABSOLUTO começa na raiz e segue um caminho até o arquivo especificado, incluindo os nomes de diretório no caminho. Um nome de caminho RELATIVO define um caminho a partir do diretório atual.
Na exclusão de um diretório, se o mesmo estiver vazio, sua entrada no diretório que o contém pode ser excluída. Se o diretório não estiver vazio, contendo vários arquivos ou subdiretórios, podemos utilizar duas técnicas: alguns sistemas não excluirão um diretório se ele não estiver vazio (tendo o usuário que excluir todos os arquivos desse diretório, executando esse processo recursivamente para os subdiretórios do diretório a ser apagado). Ou, quando for requisitada a exclusão de um diretório, todos os arquivos e subdiretórios desse diretório serão excluídos.
Com um sistema de diretórios em árvore, os usuários podem ter permissão de acesso a arquivos de outros usuários, especificando o nome do caminho.

Uma estrutura de árvore proíbe o compartilhamento de arquivos ou diretórios. Um grafo acíclico permite aos diretórios compartilharem subdiretórios e arquivos. O mesmo arquivo ou subdiretório pode estar em dois diretórios diferentes. É importante observar que um arquivo compartilhado não é o mesmo que duas cópias do arquivo. Com um arquivo compartilhado só existe UM arquivo real, de modo que quaisquer mudanças feitas por uma pessoa são visíveis imediatamente à outra.
Os arquivos e diretórios compartilhados podem ser implementados de várias maneiras, uma maneira é o LINK, que é um ponteiro para outro arquivo ou subdiretório, um link pode ser implementado como um nome de caminho absoluto ou relativo. Quando é feita uma referência a um arquivo, pesquisamos o diretório, se a entrada do diretório for um LINK, então o nome do arquivo real é incluído nas informações do link. Resolvemos o link usando o nome de caminho

para localizar o arquivo real. Outra técnica comum é duplicar todas as informações sobre eles nos diretórios que os compartilham. Assim, as duas entradas são idênticas e iguais. Um problema nisso é manter a coerencia quando um arquivo é modificado.
Uma estrutura com grafo acíclico é mais flexível do que uma estrutura de árvore simples, mas também é mais complexa. Vários problemas precisam ser considerados com cuidado, um arquivo agora pode ter vários nomes de arquivo absolutos.

Quando acrescentamos links a um diretório estruturado em árvore existente, a estrutura de árvore é destruída, resultando em uma estrautura de grafo simples. A principal vantagem de um grafo acíclico é a relativa simplicidade dos algoritmos para atravessar o grafo e determinar quando não existem mais referências a um arquivo. Queremos evitar a travessia de seções compartilhadas de um grafo acíclico duas vezes, principalmente por motivos de desempenho. Em geral precisamos usar um esquema de coleta de lixo (garbage collection) para determinar quando a última referência foi excluída e o espaço em disco pode ser realocado. A coleta de lixo envolve a travessia do sistema de arquivos inteiro, marcando tudo que pode ser acessado. Depois, uma segunda passada coleta tudo o que não está marcado na lista de espaço livre. No entanto, a coleta de lixo para um sistema de arquivos baseado em disco é muito demorada e, portanto, raramente é experimentada. A coleta de lixo só é necessária por causa dos possíveis ciclos no grafo, assim uma estrutura de grafo acíclico é muito mais fácil de trabalhar. A dificuldade é evitar ciclos à medida que novos links são acrescentados à estrutura.

IMPLEMENTAÇÕES DO DIRETÓRIO

O método mais simples de implementar um diretório é usar uma lista linear de nomes de arquivo com ponteiros para os blocos de dados. Para criar um novo arquivo primeiro temos de pesquisar o diretório para ter certeza de que não existe um arquivo com o mesmo nome. Depois incluímos uma nova entrada no final do diretório. A desvantagem real de uma lista linear de entradas de diretório é que encontrar um arquivo requer uma busca linear, as informações dos diretórios são usadas com frequencia e o usuário observará se o acesso é lento. Uma lista classificada permite uma pesquisa binária e diminui o tempo médio da pesquisa, contudo o requisito de manter a lista ordenada pode complicar o criação e a exclusão de arquivos.

Outra estrutura usada é a TABELA HASH. Uma lista linear armazena as entradas de diretório, mas uma estrutura de dados de hash também é usada. A tabela de hash apanha um valor calculado do nome do arquivo e retorna um ponteiro ao nome do arquivo na lista linear (diminuindo o tempo de busca do diretório). A inserção e a exclusão são bastante simples, embora seja necessário prever as COLISÕES (situações onde dois nomes de arquivos são traduzidos para o mesmo lugar). As dificuldades da tabela hash, são seu tamanho, em geral fixo, e a dependência da função de hash sobre esse tamanho. Como alternativa, uma tabela de hash com estouro encadeado pode ser utilizada. Cada entrada de hash pode ser uma lista interligada, em vez de um valor individual, e podemos resolver as colisões acrescentando a nova entrada à lista interligada. AS pesquisas podem ser um pouco mais lentas, pois a procura de um nome pode exigir a busca em uma lista interligada de entradas de tabela em colisão, mas será mais rápido do que uma pesquisa linear pelo diretório inteiro.

MÉTODOS DE ALOCAÇÃO

O problema principal é como alocar espaço para esses arquivos de modo que o espaço no disco seja utilizado de forma eficaz e os arquivos possam ser acessados com rapidez.

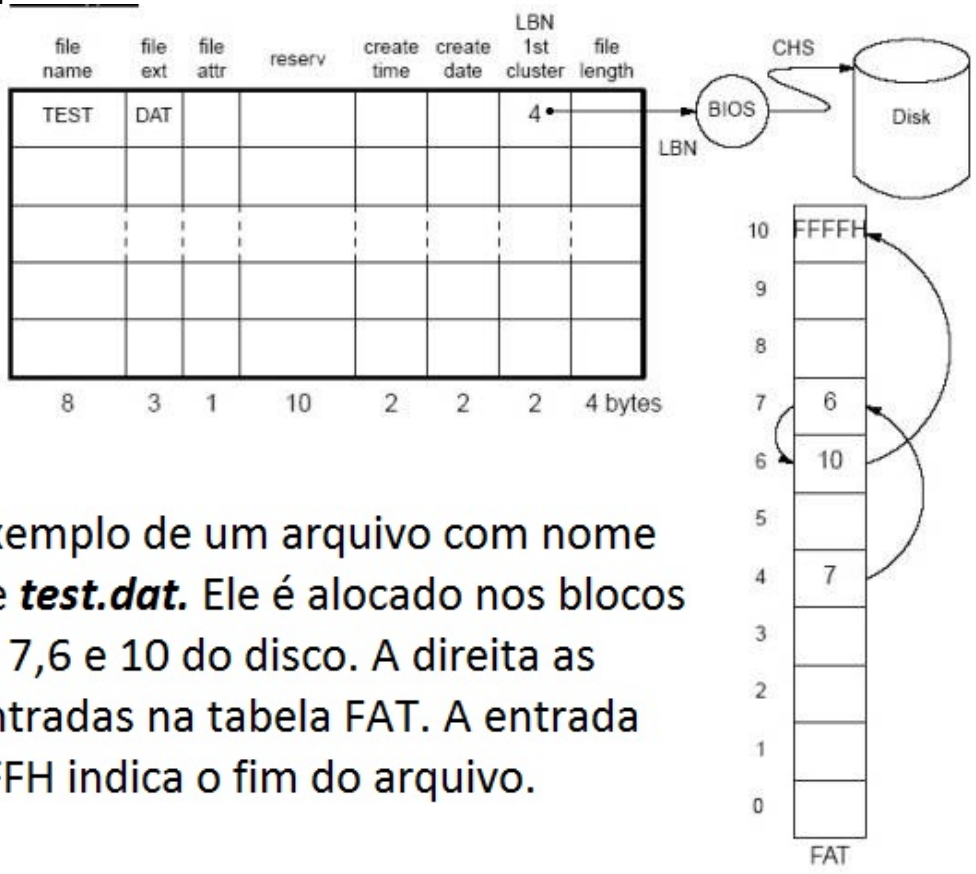
A ALOCAÇÃO CONTÍGUA requer que cada arquivo ocupe um conjunto de blocos contíguos no disco. Os endereços de disco definem uma ordenação linear no disco. A quantidade de buscas de disco exigidas para acessar arquivos alocados consecutivamente é mínima, assim como o tempo de busca quando uma busca é necessária.
A alocação contígua de um arquivo é definida pelo endereço e tamanho do disco (em unidades de bloco) do primeiro bloco. Se o arquivo ocupar n blocos de extensão e começar no local b, ele ocupará os blocos b, b+1, b+2,...,b+n-1. A entrada de diretório para cada arquivo indica o endereço do bloco inicial e o tamanho da área alocada para esse arquivo.
O acesso consecutivo é fácil, o sistema de arquivos se lembra do endereço de disco do último bloco referenciado e, quando necessário, lê o bloco seguinte. Para o acesso ao bloco i de um arquivo que começa no bloco b, podemos acessar imediatamente o bloco b+i. Assim, os acessos sequencial e direto podem ser admitidos pela alocação contígua.
Uma dificuldade é encontrar espaço para um novo arquivo. O sistema escolhido para gerenciar o espaço livre determina como essa tarefa é realizada. Os algoritmos sofrem com o problema da fragmentação externa. À medida que os arquivos são alocados e excluídos, o espaço livre em disco é dividido em pequenos pedaços. Isso se torna um problema quando o maior trecho contíguo é insuficiente para uma requisição;
Outro problema na alocação contígua é determinar quanto espaço é necessário para um arquivo. Quando o arquivo é criado, a quantidade total de espaço de que ele precisará deve ser encontrada e alocada. Se alocarmos pouco espaço, podemos descobrir que ele não pode ser estendido, logo, não podemos tornar esse arquivo maior no local. O programa do usuário pode ser terminado, com uma mensagem de erro, o usuário precisará alocar mais espaço e executar o programa novamente.
Para reduzir essas desvantagens, alguns sisops utilizam um esquema modificado de alocação contígua. Um trecho contíguo é alocado inicialmente e, depois, se esse espaço não for grande o bastante, outro trecho de espaço contíguo (uma EXTENSÃO) é adicionado. O local dos blocos de um arquivo é então registrado como um local e um contador de bloco, com um link para o primeiro bloco da próxima extensão.

A **ALOCAÇÃO INTERLIGADA** soluciona todos os problemas de alocação contígua. Com a alocação interligada cada arquivo é uma lista interligada de blocos de disco; os blocos do disco podem estar espalhados em qualquer lugar do disco. O diretório contém um ponteiro para o primeiro e último blocos do arquivo. Cada bloco contém um ponteiro para o próximo bloco.
Não existe fragmentação externa com a alocação interligada, e qualquer bloco livre na lista de espaço livre pode ser usado para satisfazer uma requisição.. O tamanho de um arquivo não precisa ser declarado quando ele é criado. Um arquivo pode continuar a crescer enquanto os blocos livres estão disponíveis. Como resultado, nunca é necessário compactar o espaço em disco.
Contudo a alocação interligada possui desvantagens, o maior problema é que ela só pode ser usada de modo eficiente para os arquivos de acesso sequencial. Para encontrar o bloco de posição i de um arquivo temos de começar no início desse arquivo e seguir os ponteiros até chegarmos ao bloco na posição i.
Outra desvantagem é o espaço exigido para os ponteiros, se um ponteiro precisa de 4 bytes de um bloco de 512 bytes, então 0,78% do disco está sendo usado para ponteiros, em vez de informação. A solução para esse problema é coletar blocos em múltiplos, chamados clusters, e alocar clusters em vez de blocos. Por exemplo, o sistema de arquivos pode definir um cluster como quatro blocos e operar sobre o disco apenas em unidades de cluster. Os ponteiros então,

utilizam uma porcentagem muito menor do espaço em disco ocupado pelo arquivo.
Outro problema da alocação interligada é a confiabilidade. Os ponteiros estão espalhados por todo o disco, interligados. O que aconteceria se um ponteiro estivesse perdido ou danificado? Uma solução parcial é usar listas duplamente interligadas e outra é armazenar o nome do arquivo e o número relativo do cloco em cada bloco; porém esses esquemas exigem ainda mais custo adicional para cada arquivo.

Uma variação da alocação interligada é o uso de uma TABELA DE ALOCAÇÃO DE ARQUIVOS (File Allocation Table - FAT). Uma seção do disco, no início de cada volume, é separada para conter a tabela. A tabela possui uma entrada para cada bloco de disco e é indexada por número de bloco. A FAT é usada de modo bastante semelhante a uma lista interligada. A entrada do diretório contém o número do primeiro bloco do arquivo. A entrada da tabela indexada por esse número de bloco contém o número do próximo bloco no arquivo. Essa cadeia continua até o último bloco, que possui um valor especial de fim de arquivo como entrada na tabela. Os blocos não-usados são indicados por um valor de tabela 0. A alocação de um novo bloco a um arquivo é uma simples questão de localizar a primeira entrada da tabela com valor 0 e substituir o valor anterior de fim de arquivo pelo endereço do novo bloco. O valor 0, em seguida, é substituído pelo valor de fim de arquivo.
Na pior das hipóteses dois movimentos da cabeça do disco ocorrem para cada um dos blocos. Um benefício é que o tempo de acesso aleatório é melhorado, pois a cabeça do disco pode encontrar o local de qualquer bloco lendo a informação na FAT.

Exemplo:



Exemplo de um arquivo com nome de ***test.dat.*** Ele é alocado nos blocos 4, 7,6 e 10 do disco. A direita as entradas na tabela FAT. A entrada FFFH indica o fim do arquivo.

A **ALOCAÇÃO INDEXADA** reune todos os ponteiros em um único local: o BLOCO DE ÍNDICE. Cada arquivo possui seu próprio bloco de índice, que é um array de endereços de bloco do disco. Para encontrar e ler o bloco i, usamos o ponteiro para a entrada de bloco de índice i. A alocação indexada admite acesso direto, sem sofrer com a fragmentação externa, pois qualquer bloco livre no disco pode satisfazer uma requisição de mais espaço. Contudo, sofre com o espaço desperdiçado, o custo adicional de ponteiro do bloco de índice em geral é maior do que o custo adicional de ponteiro da alocação interligada.

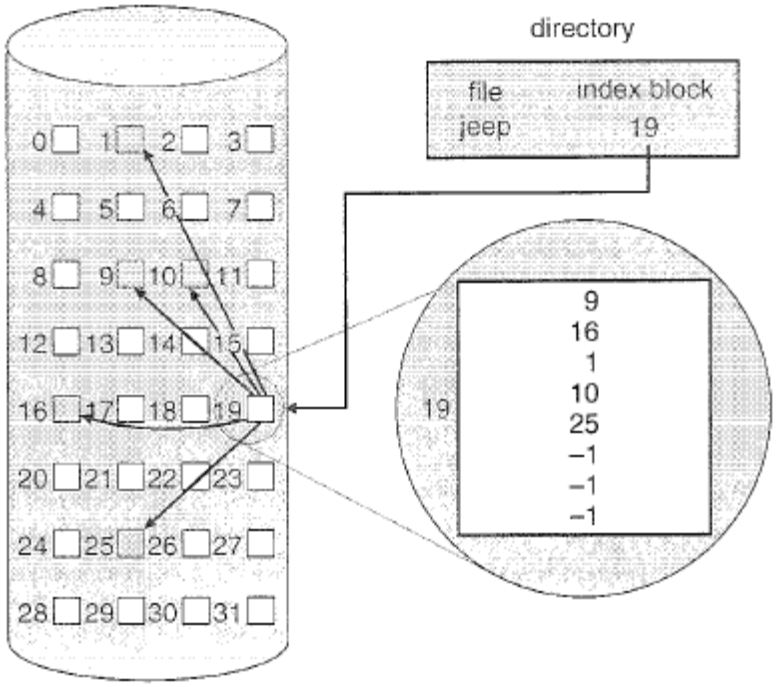


Figure 11.8 Indexed allocation of disk space.

*esquema interligado: ter um ponteiro para outro bloco de índice (caso o arquivo seja grande).
*índice multinível: usar um bloco de índice de primeiro nível para apontar para um conjunto de blocos de índice de segundo nível, que por sua vez aponta para os blocos do arquivo.
*esquema combinado:

Os esquemas de alocação indexada sofrem dos mesmos problemas de desempenho da alocação interligada, especialmente, os blocos de índice podem ser colocados em cache na memória, mas os blocos de dados podem estar espalhados por todo um volume.

Antes de requisitarmos um método de alocação precisamos determinar como os sistemas serão usados, um sistema com acesso principalmente sequencial não deve utilizar o mesmo método que um sistema com acesso principalmente aleatório. Para qualquer tipo de acesso a alocação contígua requer apenas um acesso para obter um bloco do disco. para alocação interligada, também podemos manter o endereço do próximo bloco na memória e lê-lo diretamente, esse método serve para o acesso sequencial; porém, para o acesso direto, um acesso ao bloco i poderia exigir i leituras do disco. Esse problema indica o porque de uma alocação interligada não dever ser usada para uma aplicação que exige o acesso direto. Alguns sistemas admitem os arquivos de acesso direto usando a alocação contígua e os de

acesso sequencial pela alocação interligada. Para esses sistemas, o tipo de acesso a ser feito precisa ser declarado quando o arquivo é criado. Um arquivo criado para acesso sequencial será vinculado e não poderá ser usado para acesso direto. Um arquivo criado para acesso direto será contíguo e poderá admitir o acesso direto e o acesso sequencial, mas seu tamanho máximo terá de ser declarado quando criado.
Alguns sistemas combinam a alocação contígua com a alocação indexada, usando a alocação contígua para arquivos pequenos (até três ou quatro blocos) e passando automaticamente para a alocação indexada se o arquivo crescer.

Dado a disparidade entre velocidade de CPU e disco, é válido acrescentar milhares de instruções extras ao sistema operacional para economizar apenas alguns movimentos na cabeça de leitura/escrita do disco.

GERENCIAMENTO DO ESPAÇO LIVRE
Para acompanhar o espaço livre no disco, o sistema mantém uma LISTA DE ESPAÇO LIVRE. Essa lista registra todos os blocos de disco livres (aqueles não alocados para algum arquivo ou diretório). Para criar um arquivo, pesquisamos a lista de espaço livre em busca da quantidade de espaço exigida e alocamos esse espaço para o novo arquivo. Esse espaço é então removido da lista de espaço livre. Quando o arquivo é excluído, seu espaço em disco é acrescentado à lista de espaço livre. A lista de espaço livre, apesar do nome, pode não ser implementada como uma lista.

Frequentemente a lista de espaço livre é implementada como um MAPA DE BITS ou VETOR DE BITS. Cada bloco é representado por 1 bit. Se o bloco estiver livre o bit é 1 se o bloco estiver alocado, o bit é 0. A principal vantagem é a sua simplicidade e eficiência na localização do primeiro bloco livre ou n blocos livres consecutivos no disco.
Infelizmente os vetores de bits só serão eficientes se o vetor inteiro for mantido na memória principal (e gravado no disco ocasionalmente, para as necessidades de recuperação).

LISTA INTERLIGADA: interliga todos os blocos de disco livres, mantendo um ponteiro para o primeiro bloco livre em um local especial no disco e colocando-o em cache na memória. Esse primeiro bloco contém um ponteiro para o próximo bloco do disco livre, e assim por diante. Esse esquema não é eficiente; para atravessar a lista, temos de ler cada bloco, o que requer um tempo de E/S substancial. felizmente, a travessia da lista de espaço livre não uma ação frequente.

AGRUPAMENTO: Uma modificação na técnica de espaço livre é armazenar os endereços de n blocos livres no primeiro bloco livre. Os primeiros n-1 desses blocos estão livres. O último bloco contém os endereços de outros n blocos livres e assim por diante. Os endereços de uma grande quantidade de blocos livres podem ser localizados rapidamente.

CONTAGEM: em geral vários blocos contiguos podem ser alocados ou liberados simultaneamente, em especial quando o espaço é alocado com o algoritmo de alocação contígua ou por meio do agrupamento. Assim, em vez de manter uma lista de n endereços de disco livres, podemos manter o endereço do primeiro bloco livre e o número n de blocos contíguos livres existentes após o primeiro bloco. Cada entrada na lista de espaço livre

consiste em um endereço de disco e um contador. Embora cada entrada exija mais espaço do que um endereço de disco simples, a lista geral será mais curta se o contador for maior que 1.

A maioria dos controladores de disco inclui memória local para formar um cache na placa, suficientemente grande para armazenar trilhas inteiras de cada vez. Quando uma busca é realizada, a trilha é lida para a cache do disco, começando no setor sob a cabeça do disco (reduzindo o tempo de latência). O controlador do disco, em seguida, transfere quaisquer requisições de setor para o sistema operacional. Quando os blocos passam do controlador de disco para a memória principal, o sistema operacional pode colocar os blocos no cache da memória.
Alguns sistemas mantém uma seção separada da memória principal para um cache do buffer, na qual os blocos são mantidos sob a suposição de que serão usados novamente em breve. Outros sistemas colocam dados de arquivo em cache usando um cache de página. O cache de página usa técnicas de memória virtual para colocar dados de arquivo em cache como páginas, em vez de blocos orientados ao sistema de arquivos. O caching de dados de arquivo usando endereços virtuais é muito mais eficiente do que o caching de blocos de disco físicos, pois os acesso são ligados à memória virtual, em vez de ao sistema de arquivos.

Escritas síncronas ocorrem na ordem em que o subsistema de disco as recebe, e as escritas não são colocadas em buffer. Assim, a rotina de chamada precisa esperar até os dados alcançarem a unidade de disco antes de poder prosseguir.
Em uma escrita ASSÍNCRONA, os dados são armazenados no cache, e o controle retorna a quem chamou.

Free-behind: remove uma página do buffer assim que a página seguinte é requisitada. As páginas já usadas provavelmente não serão mais usadas, e assim desperdiçam espaço no buffer.
Read ahead: uma página requisitada e várias páginas subsequentes são lidas e mantidas em cache. Essas páginas podem ser requisitadas após a pagina atual ser processada. A leitura desses dados do disco em uma transferência e sua manutenção em cache economizam um tempo considerável.

MONTAGEM DO SISTEMA DE ARQUIVOS
ASsim como um arquivo precisa ser aberto antes de ser usado, um sistema de arquivos precisa ser MONTADO antes de poder estar disponível para os processos no sistema.
O procedimento de montagem é simples. O sisop recebe o nome do dispositivo e do PONTO DE MONTAGEM - o local dentro da estrutura de arquivos onde o sistema de arquivos deve ser anexado. Normalmente um ponto de montagem é um diretório vazio. Por exemplo, em um sistema UNIX, um sistema de arquivos contendo o diretório home de um usuário poderia ser montado como /home; Após o sisop verifica se o dispositivo contém um sistema de arquivos válido. ele faz isso pedindo ao driver de dispositivo para ler o diretório do dispositivo e verificando se o diretório possui o formato esperado. Por fim, o sistema operacional observa em sua estrutura de diretórios que o sistema de arquivos está montado no ponto de montagem especificado.

Sistemas de arquivos estruturados por logs: fundamentalmente, todas as mudanças de

meta-dados são escritas em sequência em um log. Cada conjunto de operações que realiza uma tarefa específica é uma TRANSAÇÃO. Quando as mudanças são escritas nesse log, elas são consideradas confirmadas, e a chamada de sistema pode retornar ao processo do usuário, permitindo que ele continue a execução. Nesse meio tempo essas entradas de log são reproduzidas pelas estruturas reais do sistema de arquivos. Enquanto as mudanças são feitas, um ponteiro é atualizado para indicar quais ações foram concluídas e quais ainda estão incompletas. Quando uma transação confirmada inteira é concluída, ela é removida do arquivo de log, que, na realidade é um buffer circular.
O log pode estar em uma seção separada do sistema de arquivos ou em um eixo de disco separado. É mais eficiente, porém mais complexo, que o log esteja sob cabeças de leitura/escrita separadas, diminuindo assim a disputa pela cabeça e os tempos de busca.
Se um sistema falhar, o arquivo de log conterá zero ou mais transações. Quaisquer transações contidas nunca foram completadas para o sistema de arquivos, embora tenham sido confirmadas pelo sistema operacional, de modo que agora precisam ser concluídas. AS transações podem ser executadas do ponteiro até o trabalho estar completo, de modo que as estruturas do sistema de arquivos permanecem consistentes.

DISCO MAGNÉTICO
A superfície de um prato é dividida logicamente em TRILHAS circulares, que são subdivididas em SETORES. O conjunto de trilhas que estão em uma posição do braço forma um CILINDRO.

Taxa de transferência (transfer rate) é a velocidade com que os dados fluem entre a unidade e o computador.
Tempo de posicionamento (positioning time), às vezes chamado de TEMPO DE ACESSO (access time), consiste em duas partes: o tempo para mover o braço do disco até o cilindro desejado, chamado de tempo de busca (SEEK TIME), e o tempo para que o setor desejado gire até a posição da cabeça de leitura/escrita, chamado de LATÊNCIA ROTACIONAL (rotational latency).

Largura de banda do disco é o número total de bytes transferidos, dividido pelo tempo total entre a primeira requisição de serviço e o término da última transferência. Podemos melhorar tanto o tempo de acesso quanto a largura de banda escalonando o serviço das requisições de E/S de disco em uma boa ordem.

Fitas magnéticas possuem um acesso muito mais lento que um disco. São usadas para backup, de informações usadas com pouca frequencia e como um meio de transferir informações de um sistema para outro. Uma fita é mantida sobre um carretel e avança e retrocede sob uma cabeça de leitura/escrita. A movimentação até a posição correta de uma fita pode levar minutos.

As unidades de disco modernas são endereçadas como grandes arrays unidimensionais de blocos lógicos, sendo o bloco lógico a menor unidade de transferência. O array é mapeado sequencialmente nos setores do disco. O setor 0 é o primeiro setor da primeira trilha no cilindro mais externo. O mapeamento segue na sequencia por essa trilha, depois pelo restante das trilhas nesse cilindro, e depois pelo restante dos cilindros, do mais externo aos mais internos. Quanto mais afastada uma trilha tiver do centro do disco, maior é seu tamanho, de modo

que ela pode conter mais setores. A unidade aumenta sua velocidade de rotação enquanto a cabeça se move das trilhas mais externas para as mais internas, a fim de manter a mesma taxa de dados passando sob a cabeça.

Sempre que um processo precisa de E/S de/para o disco, ele emite uma chamada de sistema para o sisop, a requisição especifica várias informações:

*Se essa operação é de entrada ou saída.
*Qual é o endereço de disco para a transferência.
*Qual é o endereço de memória para a transferência.
*Qual é a quantidade de bytes a serem transferidos.

Se a unidade de disco desejada e seu controlador estiverem disponíveis, a requisição pode ser atendida imediatamente. Se a unidade ou o controlador estiver ocupado, quaisquer requisições de atendimento novas serão colocadas na fila de requisições pendentes para essa unidade.

Escalonamento FCFS: forma mais simples de escalonamento de disco, primeiro a chegar, primeiro a ser atendido. Esse algoritmo é justo mas em geral não provê o serviço mais rápido. Pode haver deslocamentos muito grandes pelo disco, o que compromete o desempenho.

Escalonamento SSTF (Shortest-Seek-Time-First): atender todas as requisições próximas à posição atual da cabeça, antes de mover a cabeça para longe, a fim de atender outras requisições. O algoritmo seleciona a requisição com o tempo de busca mínimo a partir da posição atual da cabeça. Esse algoritmo provê uma melhoria substancial no desempenho. Pode ocorrer starvation, sempre chegar requisições mais próximas e o escalonamento nunca chegar a requisições que estão longe da cabeça.

Escalonamento SCAN: o braço do disco começa em uma extremidade do disco e passa para a outra extremidade, atendendo às requisições à medida que alcança cada cilindro, até chegar à outra extremidade do disco. Na outra extremidade, a direção do movimento da cabeça é invertida, e o atendimento continua. A cabeça passa continuamente para a frente a para trás no disco. O algoritmo SCAN é as vezes chamado de algoritmo elevador, pois o braço se comporta como um elevador de um prédio, primeiro atendendo todas as requisições de subida e depois retornando para atender às requisições no outro sentido. Se uma requisição chegar logo na frente da cabeça, ela será atendida quase imediatamente, se chegar logo atrás da cabeça terá de esperar até o braço se mover até o final do disco, reverter a direção e retornar.

Escalonamento C-SCAN: o escalonamento SCAN circular (C-SCAN) é uma variante do SCAN, projetada para fornecer um tempo de espera mais uniforme. Como o SCAN, o C-SCAN move a cabeça de uma extremidade do disco para a outra, atendendo às requisições durante o caminho. No entanto, quando a cabeça atinge a outra extremidade, ela imediatamente retorna ao início do disco, sem atender a quaisquer requisições na viagem de volta. O algoritmo basicamente trata os cilindros como uma lista circular que contorna do último cilindro para o primeiro.

Escalonamento LOOK: Na prática, nem SCAN, nem C-SCAN é implementado dessa maneira. Normalmente o braço só vai até o ponto da última requisição em cada sentido. Depois ele inverte o sentido, sem ir até a extremidade do disco. Essas versões de C-SCAN e SCAN são chamadas de ESCALONAMENTO LOOK e C-LOOK, pois procuram (look for) uma requisição

antes de continuar movendo em um sentido indicado.

Um algoritmo de escalonamento de disco deve ser escrito como um módulo separado do sistema operacional, de modo a poder ser substituído por um algoritmo diferente, se necessário.

Antes de um disco poder armazenar dados, ele precisa ser dividido em setores que o controlador de disco possa ler e escrever. Esse processo é denominado formatação física. A formatação preenche o disco com uma estrutura de dados especial para cada setor. A estrutura de dados para um setor consiste em um cabeçalho, uma área de dados e um término.

Um programa de boot encontra o kernel do sistema operacional no disco, carrega o kernel na memória e salta para um endereço inicial para iniciar a execução do sistema operacional. No windows, o código de boot é colocado no primeiro setor do disco rígido (que ele chama de registro mestre de boot ou MBR). Uma partição, identificada como partição de boot, contém o sistema operacional e drivers de dispositivo. O boot começa, executando o código que é residente na ROM do sistema. Esse código instrui o sistema a ler o código de boot do MBR. Além de conter o código de boot, o MBR contém uma tabela listando as partições para o disco rígido e um flag indicando de qual partição o sistema deve ser inicializado. Quando o sistema identifica a partição do boot, ele Lê o primeiro setor dessa partição (que é chamado de setor de boot) e continua com o restante do processo de boot, que inclui a carga de vários subsistemas e serviços do sistema.

Normalmente, um ou mais setores do disco se tornam defeituosos. A maioria dos discos vem até mesmo com blocos defeituosos de fábrica. Quando é encontrado um bloco defeituoso, ele escreve um valor especial na entrada correspondente na FAT para indicar às rotinas de alocação que não usem esse bloco. Discos mais sofisticados, mantém uma lista de blocos defeituosos no controlador. A lista é inicializada durante a formatação e é atualizada durante o tempo de vida do disco. O controlador pode ser informado para substituir logicamente cada setor defeituoso por um dos setores de reserva (esse esquema é conhecido como RESERVA DE SETOR). Como alternativa para a reserva de setor, alguns controladores podem ser instruídos a substituir um bloco defeituoso por DESLIZAMENTO DE SETOR. Ele "empurra" todos os setores para o lado, a fim de colocar o bloco no setor logo em seguida ao que está com defeito.

--RAID
Agora é economicamente viável conectar uma grande quantidade de discos a um sistema de computador. ter uma grande quantidade de discos em um sistema gera oportunidades de melhorar a taxa em que os dados podem ser lidos ou escritos, se os discos forem operados em paralelo. Além do mais, essa configuração provê o potencial de melhorar a confiabilidade do armazenamento de dados, já uqe podemos armazenar informações redundantes em vários discos. ASsim, a falha em um disco não ocasiona a perda dos dados.
RAID, normalmente são usadas para resolver questões de desempenho e confiabilidade. No passado, eram vistos como uma alternativa economica para discos grandes e caros; hoje, RAIDs são usados por sua maior confiabilidade e maior taxa de transferência de dados.

O armazenamento pode ser estruturado de diversas maneiras. Um sistema pode ter discos conectados diretamente aos seus barramentos, nesse caso o sisop ou software do sistema pode implementar a funcionalidade RAID. Como alternativa, um controlador de host inteligente pode controlar vários discos conectados e pode implementar RAID sobre esses discos no hardware. Finalmente, pode-se usar um ARRAY DE ARMAZENAMENTO, ou ARRAY RAID. Um array raid é uma unidade isolada, com seu próprio controlador, cache (normalmente) e discos. Ele é conectado ao host por meio de um ou mais controladores SCSI ATA padrão ou FC. Essa configuração comum permite que qualquer sistema operacional e software sem funcionalidade raid tenah discos protegidos por RAID.

Se armazenarmos apenas uma cópia dos dados, então cada falha no disco resultará na perda de uma quantidade significativa de dados. A solução para o problema de confiabilidade é introduzir a REDUNDANCIA; armazenamos informações extras que normalmente não são necessárias, mas que podem ser usadas no caso de falha de um disco, para reconstruir as informações perdidas. Assim, mesmo que um disco falhe, os dados não serão perdidos.
A técnica mais simples (porém mais cara) de introduzir a redundância é duplicar cada disco. Essa técnica é chamada de ESPELHAMENTO (mirroring). Um disco lógico portanto consistem em dois discos físicos e cada escrita é executada nos dois discos.

A medida que os discos envelhecem, a probabilidade de falha aumenta, aumentando a chance de um segundo disco falhar enquanto o primeiro está sendo reparado. Contudo, apesar de todas essas considerações, os sistemas de disco espelhados proveem muito mais confiabilidade do que os sistemas de único disco.

No paralelismo de um sistema de disco, como atingido por meio do espelhamento, existem dois objetivos principais:
*Aumentar o throughput de múltiplos acessos pequenos (ou seja, acessos de página) pelo balanceamento de carga.
*Reduzir o tempo de resposta de acessos grandes.

Principais níveis de RAID:
*RAID nível 0: refere-se aos arrays de disco com espalhamento no nível de blocos, mas sem redundância (como esplhamento ou bits de paridade).
*RAID nível 1: refere-se ao espelhamento do disco.
*RAID nível 0+1: é comum em ambientes em que o desempenho e a confiabilidade são importantes, infelizmente, ele duplica a quantidade de discos necessária para o armazenamento, assim como o RAID 1 e, por isso, é mais dispendioso. No RAID 0+1, um conjunto de discos é espalhado e depois tudo isso é espelhado para outro grupo equivalente.
*RAID nível 1+0: em que os discos são esplhados em pares, e depois os pares de espelho resultantes são espalhados. Esse RAID possui algumas vantagens teóricas em relação ao RAID 0+1, por exemplo: se um único disco falhar no RAID0+1, o espalhamento inteiro ficará inacessível, deixando apenas o outro espalhamento disponível. Com uma falha no RAID 1+0, o único disco está indisponível, mas seu par espelhado ainda está disponível, assim como todo o restante dos discos.

O RAID de paridade é muito lento quando implementado no software, de modo que normalmente RAID 0, 1 ou 0 + 1 é utilizado.

Infelizmente um RAID nem sempre garante que os dados estão disponíveis para o sistema operacional e seus usuários. Um ponteiro para um arquivo poderia estar errado, por exemplo,

ou os ponteiros dentro da estrutura do arquivo poderiam estar errados. Escritas incompletas, se não recuperadas devidamente, podem resultar em dados corrompidos.

O RAID protege contra erros físicos da mídia, mas não outros erros de hardware ou software. Com um panorama tão grande de bugs de software e hardware, são grandes os perigos em potencial para os dados em um sistema.