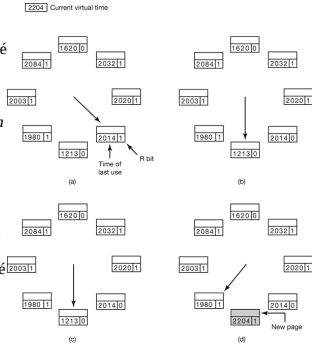
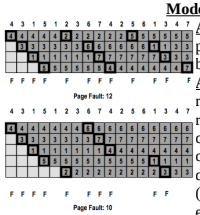
Gerenciamento de Memória

Algoritmo de substituição de página do conjunto de trabalho: Assim que a CPU tenta buscar a primeira instrução, ela detecta uma falta de página, fazendo o Sistema operacional (SO) carregar na memória a referida página que contém essa primeira instrução. Essa estratégia é denominada **pagina por demanda**, pois as páginas só são carregadas à medida que são solicitadas, e não antecipadamente.
Os processos apresentam uma propriedade denominada ***localidade de referência***, a qual diz que, durante qualquer uma das fases de sua execução, o processo só vai referenciar uma fração relativamente pequena de suas páginas.
O conjunto de páginas que um processo está atualmente usando é denominado ***conjunto de trabalho***(*working set)*. Se todo esse conjunto estiver presente na memória, o processo será executado com poucas faltas de página até mudar para outra fase de execução. Se a memória disponível for muito pequena para conter todo esse conjunto de trabalho, o processo sofrerá muitas faltas de página e será executado lentamente, pois a execução de uma instrução leva apenas uns poucos nanossegundos, mas trazer uma página do disco para a memória consome, em geral, cerca de 10 milissegundos.
Em um sistema multiprogramado, os processos são muitas vezes transferidos para disco, ou seja, todas as suas páginas são retiradas da memória, para que outros processos possam utilizar a CPU. Uma questão que surge é: o que fazer quando as páginas relativas a um processo são trazidas de volta à memória? Tecnicamente, nada precisa ser feito. O processo simplesmente causará a faltas de páginas até que seu conjunto de páginas tenha sido novamente carregado na memória. O problema é que, havendo 20, cem ou mesmo mil faltas de página toda vez que um processo é carregado, a execução se torna bastante lenta e é desperdiçado considerável tempo de CPU, pois o SO gasta alguns milissegundo de tempo de CU para processar uma falta de página.
Portanto, muitos sistemas de paginação tentam gerenciar o conjunto de trabalho de cada processo e assegurar que ele esteja presente na memória antes de o processo ser executado. Essa prática, denominada *modelo do conjunto de trabalho (working set model)*. Carregar páginas de um processo na memória *antes* de ele ser posto em execução é denomina-se também pré-paginação. Note que o conjunto de páginas se altera no tempo.
Em qualquer instante de tempo t, exite um conjunto que é constituído de todas as páginas usadas pelas k referências mais recentes à memória. Esse conjunto, *w(k, t)*, como vimos anteriormente, é o conjunto de trabalho.
O fato de a maioria dos programas acessar aleatoriamente um pequeno número de páginas e esse conjunto se alterar lentamente no tempo explica a rápida subida inicial da curva e em seguida o crescimento lento para k maiores.
Existe uma ampla faixa de valores de k para os quais o conjunto de trabalho não se altera. Para implementar esse modelo do conjunto de trabalho é necessário que o SO saiba quais são as páginas pertencentes ao conjunto de trabalho. A posse dessa informação leva imediatamente a esse possível algoritmo de substituição de página: ao ocorrer uma falta de página, encontre uma página não pertencente ao conjunto de trabalho e a remova da memória.
Para implementar um algoritmo com base no conjunto de trabalho, é preciso escolher antecipadamente um valor para k.
Obviamente, o fato de se ter uma definição operacional do conjunto de trabalho não significa que exista uma maneira eficiente de gerenciá-lo em tempo real, ou seja, durante a execução do programa. Contudo, manter um registrador de deslocamento e processá-lo a cada falta de página tem um custo proibitivo, o que faz com que essa técnica nunca seja usada.
Abandona-se a ideia de contagem de k referências à memória e usa-se, em vez disso, o tempo de execução. Podemos considerar que ele seja constituído daquelas páginas referenciadas nos últimos 110ms. Note que, para cada processo, somente seu próprio tempo de execução é considerado. Dá-se o nome de ***Tempo virtual atual*** a essa quantidade de tempo de CPU que um processo realmente empregou desde que foi iniciado.
O algoritmo de substituição de página WSClock: Baseado no algoritmo do relógio, que também usa informação do conjunto de trabalho: é o chamado WSClock.
A estrutura de dados necessárias é uma lista circular de molduras de página. Inicialmente, essa lista circular encontra-se vazia. Quando a primeira página é carregada, ela é inserida nessa lista. À medida que mais páginas são carregadas na memória, elas são também inseridas na lista para formar um anel. Cada entrada dessa lista contém o campo *Instante da última referência*.
Assim como ocorre com o algoritmo do relógio, a cada falta de página, a página que estiver sendo apontada sera examinada primeiro. Se seu bit for 1, a página foi referenciada durante a interrupção de relógio atual e assim não será candidata à remoção da memória. O bit R é então colocado em zero, o ponteiro avança para a página seguinte e o algoritmo é repetido para essa nova página.
Se sua idade for maior do que o intervalo x e se essa página estiver limpa, ela não se encontrará no conjunto de trabalho e haverá uma cópia válida em disco. Por outro lado, se a página estiver suja, ela não poderá ser reivindicada imediatamente, já que não há uma cópia válida em disco. Para evitar uma troca de processo, a escrita em disco é escalonada, mas o ponteiro é avançado e o algoritmo continua com a próxima página.



Em princípio, todas as páginas podem ser escalonadas para E/S em disco a cada volta do relógio. Para reduzir o tráfego de disco, pode-se estabelecer um limite, permitindo que um número máximo de *n* de páginas sejam reescritas em disco.
Existem duas possibilidades para o caso do ponteiro dar uma volta completa retornando ao ponto de partida:
Pelo menos uma escrita foi escalonada.
Nenhuma escrita foi escalonada.

Modelagem de algoritmo de substituição de página



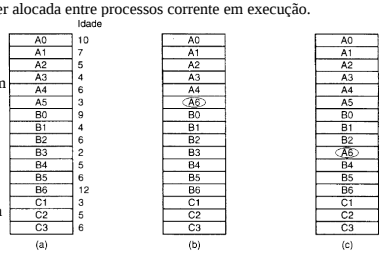
Anomalia de Belady: Pode parecer, intuitivamente, que, quanto mais página a memória possuir, menos faltas de páginas o programa terá. É bastante surpreendente que isso nem sempre é verdadeiro.
Algoritmo de pilha: Sabemos que todo processo gera uma sequência de referência à memória durante sua execução e que cada uma dessas referências corresponde a uma página virtual específica. Assim, conceitualmente, um acesso à memória alocada a um processo pode ser caracterizado por uma lista (ordenada) de números de página. Essa lista é denominada ***cadeia de referências*** e desempenha um papel centra na teoria. (com vários processos seria necessário levar em consideração o entrelaçamento de suas cadeias de referência devido à multiprogramação).
Um sistema de referência do processo pode ser caracterizado por três itens:

1. A cadeia de referências do processo em execução.
2. O algoritmo de substituição de páginas.

O número de molduras de página disponível na memória, m. Ao começar a execução, o processo passa a emitir, uma a uma, as referências às páginas da cadeia de referência. A cada referência, o interpretador verifica se a página se encontra presente na memória.
Primeiramente, quando uma página é referenciada, ela é sempre movida para o topo de M. Em segundo lugar, se a página referenciada já estiver em M, todas as páginas acima dela serão deslocadas de uma posição para baixo. Em terceiro lugar, as páginas que estão abaixo da página referenciada não são movidas.

$$M(m,r) \subseteq M(m+1,r)$$

onde m varia de acordo com as molduras de página e r é um índice da cadeia de referências. Isso quer dizer que o conjunto de páginas inserido na parte superior de M, para uma memória com m molduras de página depois de r referências à memória, também será incluído em M para uma memória com m + 1 molduras de página. Em outras palavras, se aumentarmos o tamanho da memória em uma moldura de página e reexecutarmos o processo, em cada ponto durante a execução, todas as páginas que estiveram presentes na memória na primeira execução estarão também presentes na segunda, juntamente com uma página adicional.
A cadeia de distâncias: Uma referência a uma página será de agora em diante denotada pela distância do topo da pilha até a posição onde a página referenciada estava localizada.
Note que a cadeia de distâncias depende não somente da cadeia de referências, mas também do algoritmo de paginação. As propriedades estatísticas da cadeia de distâncias representam um grande impacto no desempenho dos algoritmos.
A previsão de frequência de falta de página: O algorimo começa varrendo a cadeia de distâncias, págia a página. Ele controla o número de vezes que o 1 ocorre, o número de vezes que o 2 ocorre e assim por diante.
Questões de projeto para sistemas de paginação
Política de alocação local versus global: A maior questão associada a escolha do algoritmo de substituição de páginas consiste em como a memória deve ser alocada entre processos corrente em execução.
Algoritmos de substituição local alocam uma fração fixa de memória para cada processo. Algoritmos de substituição de página global alocam moldura de página entre os processos em execução.
Em geral, os algoritmos globais funcionam melhor, especialmente quando o tamanho do conjunto de trabalho varia durante o tempo de vida de um processo. Se um algoritmo local é usado e o conjunto de trabalho cresce durante a execução do processo, uma paginação excessiva (*thrashing*) pode ocorrer mesmo que existam muitas molduras de página disponível na memória. Se o conjunto de trabalho diminuir durante a execução do processo, os algoritmos locais desperdiçam memória. Se um algoritmo global é usado, o sistema deve decidir continuamente quantas molduras de página alocar para cada processo. Uma das maneiras de fazer isso é o monitoramento do tamanho do conjunto de trabalho desse processo, conforme indicado pelos bits de envelhecimento (*aging*), mas essa estratégia não evita necessariamente a paginação excessiva. O conjunto de trabalho pode variar de tamanho em questão de microssegundos, enquanto os campos de envelhecimento representam uma medida rudimentar estendida a um número de interrupções de relógio.



Outra prática se baseia em um algoritmo de alocação de molduras de página para processos. Uma possibilidade é determinar periodicamente o número de processos em execução e alocar para cada processo o mesmo úmero de moldura de página. As molduras que sobram vão para uma área comum (*pool*) para serem usadas na ocorrência de falta de página.
Parece razoável alocar para cada processo um número mínimo de molduras de página de modo que ele possa ser executado, não importando o quão pequeno seja.
Se um algoritmo global for usado, pode ser possível iniciar cada processo com um número de molduras de ´agina proporcional ao tamanho do processo, mas a alocação tem de ser atualizada dinamicamente durante a execução do processo. PFF (*page fault frequency*) é um algoritmo que informa quando aumentar ou diminuir a alocação de página de um processo, mas nada diz acerca de quais páginas substituir quando ocorre flata de página.
Medir a frequência de faltas de página é direto: basta contar o número de faltas por segundo e depois calcular a frequência média de faltas por segundo. Em seguida, para cada segundo, somar à média existente – ou seja, à frequência de faltas de página atual – o número de faltas ocorridas nesse novo segundo e em seguida dividir por dois a fim de obter a nova média de faltas – ou seja, a nova frequência de faltas de página atual.
Os algoritmos de conjunto de trabalho e o *WSClock* referem-se a um processo específico e, portanto, devem ser aplicados em um determinado contexto.
Controle de carga: Na verdade, sempre que os conjuntos de trabalho de todos os processos combinados excedem a capacidade da memória, pode-se esperar a ocorrência de paginação excessiva. A única solução possível é livra-se temporariamente de alguns dos processos.
Para reduzir o número de processos que competem por memória, levam-se alguns deles para disco e libera-se a memória a eles alocada. Se a paginação excessiva acabar, o sistema pode continuar a execução durante um certo tempo. Se a paginação excessiva não acabar, outro processo terá de ser levado para disco e assim por diante, até que a paginação excessiva cesse. A diferença é que agora a troca de processos é usada para reduzir a demanda potencial por memória, em vez de reivindicar blocas dela para uso imediato.
Tamanho da página: Há dois argumentos a favor de um tamanho pequeno de página. O primeiro: é provável que um segmento de código, dados ou pilha escolhido aleatoriamente não ocupe um número inteiro de páginas. Em média, metade da última página permanecerá vazia e, portanto, esse espaço será desperdiçado. Esse desperdício é denominado *fragmentação interna*. Com *n* segmentos na memória e um tamanho de página de *p* bytes, *np*/2 bytes serão perdidos com a fragmentação interna. O segundo argumento a favor de um tamanho reduzido, baseia-se que em geral, tamanho grandes de página farão com que partes do programa não usadas ocupem desnecessariamente a memória.
Por outro lado, páginas pequenas implicam muitas páginas e, consequentemente, grandes tabelas de páginas. As transferências entre memória e disco são geralmente de uma página por vez. A maior parte do tempo é gasta no posicionamento da cabeça de leitura/gravação na trilha correta e no tempo de rotação necessário para que a cabeça de leitura/gravação atinja o setor correto, de modo que se gasta muito mais tempo na transferências de páginas pequenas do de páginas grandes.
Em algumas máquinas, a tabela de páginas deve ser carregada em registradores de hardware sempre que a CPU chavear de um processo para outro. Assim, nessas máquinas, o tempo necessário para carregar os registradores com a tabela de página aumenta à media que se diminua o tamanho da página.
Espaços separados de instruções e dados: Consiste em espaços de endereçamento separados para instruções (código do programa) e dados. Esses espaços são denominados, respectivamente, espaço I e espaço D.
Em um computador com esse projeto, ambos os espaços de endereçamento podem ser paginados, independentemente um do outro. Cada um deles possui sua própria tabela de páginas, que contém mapeamento individual de páginas virtuais para molduras de página física.
Páginas compartilhadas: É nitidamente mais eficiente compartilhar páginas para evitar a situação de existirem duas cópias ou mais da mesma página presente na memória. Um problema é que nem todas as páginas são compartilháveis. Em particular, as páginas somente de leitura – como as que contêm código de programa – são compartilháveis, mas páginas com dados alteráveis durante a execução não o são.
Se o sistema suporta os espaços I e D, o compartilhamento de programas será obtido de modo relativamente direto. Em geral, em uma implementação que suporte compartilhamento desse tipo, tabelas de página são estruturas de dados independentes da tabela de processos. Cada processo tem assim dois ponteiros em sua tabela de processos: um aponta para a tabela de páginas do espaço I e outro aponta para a tabela de páginas do espaço D.
Quando dois ou mais processos compartilham o mesmo código, um problema ocorre com as páginas compartilhadas. Suponha que os processos A e B estejam ambos executando o editor e compartilhando suas páginas. Se o escalonador decidir remover A da memória, descartando todas as suas páginas e carregando as molduras de páginas vazias com outro programa, ele leva o processo B a causar muitas faltas de páginas até que suas páginas estejam novamente presentes na memória.
Em geral, é muito trabalhoso pesquisar todas as tabelas de páginas para descobrir se uma determinada página é compartilhadas, de modo que são necessárias estruturas de dados especiais para manter o controle das páginas compartilhadas.