1) PROGRAMA X PROCESSO

Programa é um Arquivo executável. Processo é um Programa carregado na memória p/ execução Partes do processo.

2) PROBLEMA DA AMARRAÇÃO DE ENDEREÇO

O problema da amarração de endereço é: **Como conseguir executar um programa independente do intervalo de endereço no qual ele será carregado?**

Isto acontece porque um processo nem sempre é carregado na posição zero da memória. As soluções para este problema são:

- Correção de endereços em tempo de carga (ou Amarração em tempo de carga);
- Registrador de base (ou Endereços Relativos à Base);
- Endereços Relativos à Instrução corrente;
- Segmentação;
- Paginação.

3) CORREÇÃO DE ENDEREÇOS EM TEMPO DE CARGA

O montador ou compilador gera código imaginando que o programa vai ser colocado no endereço 0 da memória física. Ele indica em uma tabela adicional do arquivo executável os endereços que precisam ser corrigidos. O sistema operacional antes de colocar o arquivo na memória percorre esta tabela e corrige os endereços sinalizados.

O sistema operacional sabe onde será carregado o arquivo que sai do disco e será colocado na memória para execução. Ele vai a todos os endereços do programa e soma o endereço inicial daquele programa, conforme mostra o exemplo de carregamento do processo 2 abaixo:

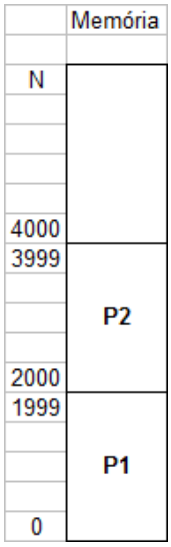
Código em Linguagem de Alto Nível

A = A + 10;

Código em Linguagem de Máquina do P1

Mov EAX, [500]
Add EAX, 10
Mov [500], EAX

Obs: Endereço Inicial do programa = 0 (Não temos nenhum problema nesse caso, pois a memoria estava vazia e o processo 1 foi carregado realmente no endereço 0).



Código em Linguagem de Alto Nível

A = A + 10;

Código em Linguagem de Máquina do P2

Mov EAX, [2500]
Add EAX, 10
Mov [2500], EAX

Obs.: Endereço Inicial do programa = 2000. Logo 2000+500 = 2500 (Endereço corrigido pelo SO ao carregar o programa para memoria. Desta forma o processo 2 funcionará sem nenhum problema).

O programa executável é formado por código mais informações de controle que o sistema operacional usa para a correção de endereço. Pois, o sistema operacional precisa de uma referência de quais instruções ele precisa alterar, essas informações de controle dizem quais são as rotinas que precisam ser alteradas.

Um exemplo de sistema operacional que usa este tipo de solução é o MS-DOS e Windows 16 bits para arquivos executáveis (.exe). As versões do Windows 32 bits e superiores não são mais assim. Esta solução não precisa de nenhum mecanismo de HW para ser implementada e também é válida quando o processo está divido em áreas na memoria.

Vantagens da Correção De Endereços Em Tempo De Carga

- Conseguir colocar dois ou mais processos na memória física.
-
-

Desvantagens da Correção De Endereços Em Tempo De Carga

- Queda de desempenho, pois o SO é que faz a correção dos endereços.
-
-

4) REGISTRADOR DE BASE

Nesta solução o HW soma ao Endereço que precisa ser corrigido o Registrador de base do processo. O SO precisa colocar o valor correto no Registrado de base do processo senão não funcionará.

Exemplo: Instrução compilada com o endereço 0 (Padrão de compilação): MOV EAX, [500]

Registrador base do Processo 2 tem o valor 2000. Logo, o HW corrige a instrução para MOV EAX, [2500].

Vantagens do First Fit

-
-
-

Desvantagens do First Fit

-
-
-

Observação:

- Na Correção de endereços em tempo de carga é o SO que faz a soma e altera o endereço. No Registrador de Base é o HW que faz a soma a cada instrução executada, mas apesar da soma feita pelo HW ser muito mais rápida do que a soma feita pelo SO. Ambas as soluções funcionavam perfeitamente na época e tinham desempenho equivalente.
- O Registrador de Base é onde o processo inicia.

5) ENDEREÇOS RELATIVOS À INSTRUÇÃO CORRENTE

São endereços que contam a partir da própria instrução, ou seja, soma ao endereço da instrução o que falta para chegar ao endereço que se deseja. Conforme mostra o exemplo a seguir:

```
If B <> 0 then A:= A + 10;
```

Variável	B	A
Endereço	400	500

Código em Linguagem de Máquina

```
100    MOV EBX, [400]
106    CMP EBX, 0
112    JZ 134
116    MOV EAX, [500]
122    ADD EAX, 10
128    MOV [500], EAX
134    ~~~
```

Código em Linguagem de Máquina utilizando Endereço Relativo à Instrução Corrente

```
100    MOV EBX, [+300]
106    CMP EBX, 0
112    JZ +22
116    MOV EAX, [+384]
122    ADD EAX, 10
128    MOV [+372], EAX
134    ~~~
```

Os endereços contidos nas instruções são gerados em tempo de carga. Essa técnica não pode ser utilizada em processos que estejam fragmentados na memória. Na CPU da Intel só é possível usar endereço relativo à instrução corrente nas instruções de salto. Se o endereço base mudar tem impacto.

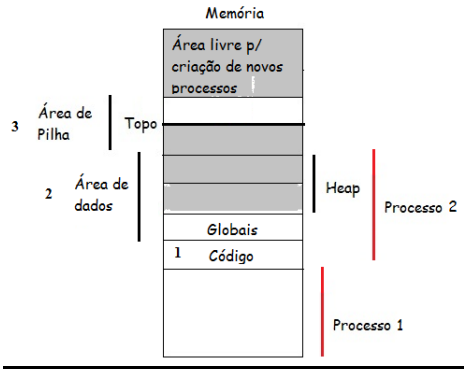
Vantagens dos Endereços Relativos À Instrução Corrente

-
-
-

Desvantagens dos Endereços Relativos À Instrução Corrente

-
-
-

6) ÁREAS DE MEMÓRIA DE UM PROCESSO



As Áreas de memória de um Processo no Unix são:

1) Área de Código;

2) Área de Dados (contém as Variáveis Globais e as Variáveis Dinâmicas (Heap));

3) Área de Pilha (contém as Variáveis Locais).

Um processo possui três áreas de memória distintas conforme sinalizado na figura anterior. É importante lembrar que tanto a Heap quanto a Pilha possuem dentro delas áreas livres para alocação de variáveis locais (no caso a Pilha) e variáveis dinâmicas (no caso a Heap).

Mas, além das áreas livres internas no processo existem as áreas livres externas ao processo. As áreas livres externas ao processo servem para alocação de novos processos. As áreas livres internas ao processo são para as variáveis locais e dinâmicas que o processo vai precisar. Assim é a organização dos processos na memória. Este exemplo fala da alocação dos processos de forma continua é a forma mais fácil de fazer e mais simples.

Porém, as áreas de memória do processo Código, Pilha e Dados podem ser alocadas separadamente na memoria, pois assim é mais fácil encontrar uma área livre. Se existe uma regra que obrigue as áreas do processo serem alocadas juntas isto restringe a alocação.

Por exemplo, queremos alocar um processo de 300 KB é mais fácil encontrar três pedaços de 100 KB de área de memória livre do que um único espaço de 300 KB na memória. Então exigir que as três áreas do processo sejam alocadas juntas é uma desvantagem, pois dificulta a alocação de processos.

Outra vantagem de alocar as áreas do processo separadas é que se tem mais chance de crescer estas áreas quando necessário e se tiver área livre.

Existem três formas de se armazenar a variável dentro das área do processo: Global, Local e Dinâmica. Os Códigos e Dados são alocados quando o processo começa a executar. Dados locais e variáveis dinâmicas vão sendo alocados conforme o programa executa.

A Diferença entre variáveis locais e dinâmicas, é que as variáveis locais têm um sequenciamento muito claro de alocação e desalocação quando a rotina começa a rodar são alocadas todas as variáveis locais para essa rotina, quando a rotina acaba de rodar são desalocados na mesma ordem em que foram alocadas. Nas variáveis dinâmicas o programador pode escolher a ordem de desalocação, não seguem a ordem de pilha como nas locais. Não dá para usar uma pilha para alocar variáveis dinâmicas.

A grande diferença entre elas é o sequenciamento, pois a Pilha tem uma sequencia para colocar e tirar as variáveis enquanto que a Heap não tem nenhuma sequencia se coloca e tira as variáveis em qualquer ordem.

ÁREA DE CÓDIGO

Responsável por guardar o código executável do programa. Esta área sofre alteração? É modificada? Geralmente não, pois o hardware não permita que o processo (ou outro processo) faça modificação na área de código. Se o processo tentar fazer é abortado.

No entanto, às vezes durante a execução do programa novos códigos são gerados e precisam ser incorporados a área de código, mas isso não é muito comum.

ÁREA DE DADOS

Responsável por guardar as variáveis globais e as variáveis dinâmicas (que ficam na Heap). Conforme o programa vai sendo executado as variáveis dinâmicas vão sendo alocadas e desalocadas. Esta área sofre alteração? É modificada? Sim, pois contém as variáveis que serão utilizadas pelo processo na sua execução.

Se o programa utilizar muitas variáveis dinâmicas, em algum momento, é possível que não tenha mais espaço para alocar essas variáveis, nesse caso, pode abortar ou tentar crescer o tamanho da área, ou seja, o SO pode tentar crescer a heap. Quem sabe se a heap tem ou não espaço é o processo, o SO não se envolve. No Windows pode ter mais de uma heap.

VARIÁVEIS GLOBAIS

As variáveis Globais existem o tempo todo no processo que está em execução e geralmente qualquer parte do processo consegue usar, alterar e consultar esta variável. Quando o programa começa ela passa existir e quando o programa termina ela deixa de existir.

VARIÁVEIS DINÂMICAS

As variáveis Dinâmicas são criadas pela "vontade" do programador. Ele escreve um comando na linguagem para criar a variável. Só depois que este comando é executado que se pode utilizar a variável criada. Elas são alocadas utilizando a instrução NEW em Pascal e MALLOC em C. O fato de o programador ficar com esta responsabilidade cria complicações, pois o compilador não sabe quando essas variáveis serão alocadas e desalocadas.

A Heap não tem ordem correta para alocação das variáveis, pois depende do programador. Ele vai determinar quando cria e quando destrói a variável dinâmica.

A variável dinâmica normalmente é usada em linguagens de programação mais sofisticadas. Em linguagem orientada a objetos tipicamente um objeto é uma variável dinâmica.

As vezes se confundi variável dinâmica com variável ponteiro. P é variável ponteiro que aponta para um endereço, ou seja, guarda o endereço de memória da variável. P dinâmico é local fica na Pilha. O comando new cria a variável dinâmica que não está na Pilha e faz a variável local que é o P apontar para a variável dinâmica.

Digamos que na Pilha a variável local P guarde o endereço 2000. O comando New não apenas reserva uma área da memória na Heap para variável dinâmica, mas também atribui o valor desta variável ao ponteiro P. O que é atribuir o valor da variável ao ponteiro, ou seja, coloca o endereço para variável. A variável dinâmica não tem nome. O ponteiro é que aponta para esta variável dinâmica que tem nome.

Ao utilizar um comando de alocação de memória, uma variável local (P) aponta para uma variável dinâmica contida na área de HEAP. P é uma variável local armazenada na Pilha.

A HEAP aloca as variáveis a execução de comandos do tipo NEW, mas só desloca aos comandos DISPODE. Logo, é possível que haja espaços livres na HEAP, no meio da HEAP alocada. O código do comando de alocação é quem percebe que a HEAP está cheia. No Pascal, quando se tenta alocar uma variável na HEAP cheia, a rotina de alocação percebe a exaustão da HEAP e aborta o processo.

No Unix, quando a HEAP esta cheia ou com espaços muito pequenos para alocação de variáveis dinâmicas, o processo faz uma chamada ao sistema para aumentar a área de dados. A área de HEAP cresce para cima, ela não cresce invadindo a área de variáveis globais.

No Windows 3.1, se a HEAP está cheia pode-se alocar outra área de dados para servir como uma segunda HEAP. Os dados na HEAP são alocados numa certa sequência, porém, nem sempre as variáveis ficam alocadas sequencialmente.

Área de Pilha

Responsável por guardar as variáveis locais. Esta área sofre modificação, pois as varáveis locais podem ser modificadas. A área de pilha é alterável. Nos SO's mais sofisticados (Unix), antes de abortar um processo o SO tenta aumentar a área da pilha, se conseguir crescer não aborta o processo.

Se uma rotina tem muita recursão, pode acabar com o espaço na pilha, causando o estouro da pilha também chamado de stack overflow. A maioria dos SOs abortam o processo – ex.: Windows 3.1. O Unix tenta aumentar o tamanho da pilha ao perceber que o espaço disponível da pilha chegou ao fim.

VARIÁVEIS LOCAIS

As Variáveis Locais só podem ser usadas dentro de uma sub-rotina. Elas são declaradas em uma sub-rotina, por exemplo, uma função. O problema maior que ocorre com as Variáveis Locais está dentro de funções recursivas.

A remoção é a complicação desta alocação de variável local em uma função recursiva, pois não existe uma única variável n na memória existem vários n. Para resolver este problema é utilizada uma pilha de execução. Na pilha de execução temos o seguinte funcionamento, supondo n = 4.

Aloca-se no sentido de cima para baixo a variável n. Assim existem quatro variáveis de nome n na memória criada a cada chamada da função recursiva: n = 4, n = 3, n = 2, n = 1. Quando desaloca na ordem inversa da alocação, de baixo para cima, é que serão feitos os cálculos do fatorial: n = 1, n = 2, n = 3, n = 4.

As Variáveis Locais possuem algumas particularidades e se assemelha muito ao parâmetro de uma função. A variável local é diferente da variável Global. A variável Global só tem uma para o programa inteiro. A local não para a mesma sub-rotina tem várias variáveis na memória com valores diferentes.

No entanto, em um determinado momento de tempo somente uma variável estará sendo utilizada. É diferente a forma de se guardar as variáveis Globais e Locais. Variáveis Locais tem mais de uma à medida que necessite, por exemplo, se o N for igual a 100. Teríamos 100 variáveis Locais na memória. Então não se sabe a priori quantas variáveis locais serão necessárias. Este número é descoberto na execução.

A Variável Global é criada no início do programa e a Variável Local é criada durante a execução do programa. Enquanto que a Variável Local é criada dinamicamente depois que a rotina acaba as variáveis são destruídas dinamicamente deixando de ocupar espaço na memória.

A primeira variável local a ser destruída e a última que foi criada e a última a ser destruída é a primeira que foi criada. A variável local é criada e destruída no momento da execução. A variável local ativa é sempre a última da execução no momento.

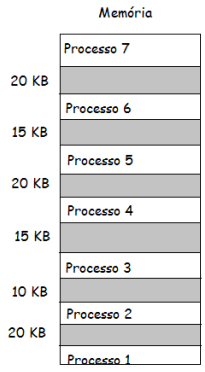
A pilha é a estrutura de dados perfeita para lógica de criação e destruição de variáveis locais. A última coisa a empilhar é a primeira a desempilhar. A primeira coisa a empilhar é a última a desempilhar.

Se aloca decrementando o topo e desaloca incrementando o topo. A ideia de pilha para o processo poder executar é uma coisa tão importante e fundamental para as variáveis locais que as CPUs tem instruções especificas para trabalhar com as pilhas isto por causa das variáveis locais.

Na área de pilha, o topo vai consumindo espaços livres para armazenar dados, até chegar ao limite inferior da área de pilha. Quando isto acontece, o HW gera uma interrupção, e podem ocorrer 2 eventos de acordo com o SO: Abortamento do processo ou/e deixar o topo continuar descendo para expansão da área de pilha. Essa expansão só é possível se, abaixo da área de pilha, houver área livre. A pilha, na CPU Intel, cresce para baixo.

O 2 evento é a expansão da área de pilha que é a copia da área de pilha para uma área livre, mas depende se o HW terá essa capacidade, pois isso acarreta uma re-amarração dos endereços em tempo de execução. É possível fazer essa re-amarração alterando-se o valor dos registradores base, como na CPU Intel para que as instruções passem a acessar os novos endereços, sem que o processo seja incomodado.

7) FRAGMENTAÇÃO DA MEMÓRIA



Os espaços de área livre que sobram ao alocar um processo na memória são chamados de Fragmentação de Memória. Conforme mostra as áreas em cinza na Figura ao lado.

Isso é um problema gerado ao alocar memória para os processos. Temos duas soluções para esse problema:

1) Tentar evitar que a fragmentação ocorra;
2) Utilizar compactação.

Veremos estas soluções a seguir em maiores detalhes.

8) SOLUÇÕES PARA FRAGMENTAÇÃO DA MEMÓRIA

1ª solução: Tentar evitar que a fragmentação ocorra

Para isso utiliza-se um algoritmo de alocação de memória para um processo de forma que as áreas livres sejam poucas e as menores possíveis.

2ª solução: Compactação

Trata-se de deslocar as áreas livres e ocupadas de forma que as áreas livres fiquem todas juntas.

A meta é trocar de posição o conteúdo da memória para reunir toda a memória livre em um grande bloco. Se a relocação é estática e for feita no momento de montagem ou carga, a compactação não poderá ser feita. A compactação só será possível se a relocação for dinâmica e for feita em tempo de execução.

Quando a compactação é possível, devemos determinar o seu custo. O algoritmo de compactação mais simples consiste em mover todos os processos em direção a um lado da memória; todos os blocos livres se movem na outra direção, gerando um grande bloco de memória disponível.

Outras soluções possíveis para o problema de fragmentação externa é permitir que o espaço de endereçamento lógico de um processo seja não contíguo, possibilitando que um processo receba memória física onde ela estiver disponível: Paginação e Segmentação.

9) ALGORITMOS DE ALOCAÇÃO

O problema é que tenho mais de uma área livre. Em qual área livre irei usar para alocar o processo? Tem vários algoritmos de alocação. Veremos alguns tipos de algoritmos de alocação.

FIRST FIT

Aloca o bloco na primeira área livre que encontrar. Não procura áreas livres na memória. Assim é mais rápido. Porém gera áreas livres de tamanho variado na memória. Começa a procurar no espaço seguinte à última alocação. Por exemplo, temos três áreas livres: 100 KB, 80 KB e 200 KB e precisamos alocar memoria para um processo 5 de 80 KB. Utilizando o algoritmo de alocação First Fit este processo é alocado no primeiro lugar que encontrar de área livre que caiba o processo (ou uma das áreas do processo). Então alocará na área livre de 100 KB. Sobrará 20 KB de área livre que não será utilizada. Isto é um desperdício de memória chamado de Fragmentação Externa.

Vantagens do First Fit

-
-
-

Desvantagens do First Fit

-
-
-

BEST FIT

Aloca o bloco na área de tamanho mais próximo ao tamanho do bloco que se quer inserir na memória. Este algoritmo pesquisa entre as áreas livres de memória o espaço mais adequado para alocar o processo. Por exemplo, temos três áreas livres: 100 KB, 80 KB e 200 KB e precisamos alocar o processo de 80 KB. Então este processo será alocado no espaço de 80 KB, pois é o ideal para ele. Neste caso, não tem Fragmentação. Mas, em outro caso pode ter Fragmentação, por exemplo, um processo de 90 Kb este algoritmo alocaria em 100 Kb desperdiçando 10 Kb.

Vantagens do Best Fit

-
-
-

Desvantagens do Best Fit

-
-
-

WORST FIT

Aloca o bloco na área de tamanho exato se existir. Se não existir uma área de tamanho exato então aloca a maior área encontrada. Começa a procurar área livre no início da memória. Por exemplo, temos três áreas livres: 100 KB, 80 KB e 200 KB e precisamos alocar o processo de 60 KB. Como não existe uma área de memória com este tamanho exato. O Worst Fit aloca a maior área livre de memória encontrada que é de 200 KB sobrando 140 KB de área livre.

O Worst Fit não gera Fragmentação porque quando ele não encontra uma área de memória de tamanho exato ele aloca a maior área e o que sobra é um tamanho grande. Embora pareça ser bom ele não é tão bom assim, pois se tivermos um processo de 150 KB poderia ser alocado na área livre de 200 KB e sobraria 50 KB, mas como o Worst Fit já matou a área livre de 200 KB (a área maior) o processo não poderá ser alocado neste espaço. E o Worst Fit é pior para este caso.

Vantagens do Worst Fit

-
-
-

Desvantagens do Worst Fit

-
-
-

Observações:

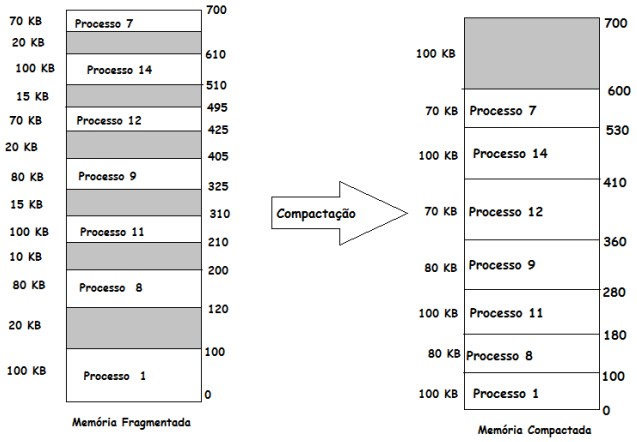
- Não tem como dizer de forma eficiente qual destes algoritmos é o melhor. O que pode se dizer é que o First Fit é o mais rápido para executar e implementar do que os outros algoritmos Best e Worst. Todos os algoritmos geram Fragmentação.
- No caso de alocar o processo por áreas separadas se tem o mesmo problema e utiliza se os mesmos algoritmos de alocação. O mesmo ocorre dentro da área de Heap, pois tem espaço livre e alocado. Na Heap é alocado as variáveis dinâmicas, mas temos o mesmo problema de alocar memória só que o espaço é menor.
- Quem controla o espaço da Heap é a própria linguagem de programação e não o Sistema Operacional. O Windows até tem suporte para isto, mas normalmente é a própria linguagem de programação que gera o código de alocação executa internamente o algoritmo segundo o fabricante da linguagem escolheu para implementar.

10) COMPACTAÇÃO

Conforme mostra a figura abaixo, a compactação junta todos os processos e toda área livre na parte de baixo da memória e acima junta todas as áreas livres. A compactação tem um problema que é a mudança dos endereços dos processos e das suas respectivas áreas. É o hardware que permite a compactação da memória. E esta compactação é acionada quando um processo deseja ocupar a memória e não tem área disponível, o SO toma a iniciativa da compactação da memória.

O mesmo problema ocorre para alocação de variáveis dinâmicas dentro de área de heap de um processo. E como solução é compactar a memória. Neste caso, para correção dos endereços tem que ser possível saber onde estão os ponteiros das variáveis dinâmicas. O Java consegue fazer isto, mas o C, Pascal não conseguem.

Digamos que antes da compactação uma variável global de um processo estava no endereço 65000 após a compactação seu endereço foi modificado. Então a mera compactação simples não vai funcionar, pois não adianta fazer somente a compactação. Precisamos alterar os ponteiros para os novos endereços de memória. Logo, a compactação exige que se tenha mecanismos para modificar os endereços senão dará problemas.



O sistema operacional que usa endereço absoluto tem condição de corrigir o endereço perdido apontando para o endereço de carga, mas o problema é que os sistemas operacionais de hoje em dia não sabem dentro da área de memória do processo onde está a variável ponteiro. Então em um esquema como este a compactação não pode ser usada, pois o sistema operacional não sabe corrigir a variável de ponteiro.

Para compactação poder ser implementada tem que utilizar Registrador de Base, pois o endereço das variáveis inclusive a variável ponteiro será feito a partir do registrador de base. Durante a execução o valor de deslocamento é somado ao valor do Registrador de Base gerando o novo endereço.

Vantagens da Compactação

-
-
-

Desvantagens da Compactação

-
-
-

Observação

- Algumas CPUs permitem várias bases. A Intel permite três bases. Uma base para área de código, uma base para área de dados e outra base para área de Heap. Tem três registradores de base diferentes um para cada área.

11) FALTA DE MEMÓRIA

É quando o usuário quer executar processos que não cabem na memória da máquina. Isto pode acontecer quando:

- Um processo executado for maior do que a memória da máquina;
- Ou a soma dos tamanhos de cada processo que se quer executar na máquina é maior que a memória desta máquina;
- O tamanho total dos processos é maior que o tamanho da memória que está disponível.

12) SOLUÇÕES PARA FALTA DE MEMÓRIA

A falta de memória é um problema de memória insuficiente mais comum no passado e hoje em dia acontece muito menos. Tratamento das imagens digitais edição de vídeo precisa de muita memória.

Sendo a edição de vídeo que mais consume memória hoje em dia. Neste caso a memória pode encher e é preciso ter um mecanismo para não dá erro para o usuário.

Resolve-se o problema de falta de memória através dos mecanismos abaixo que usa o disco ou procura gastar menos memória.

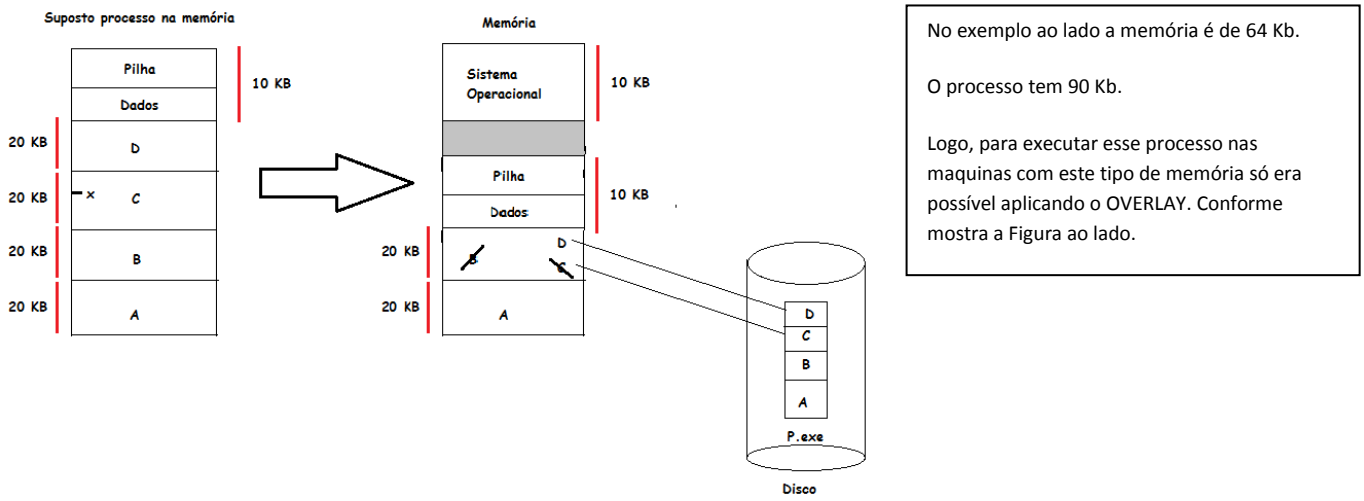
- OVERLAY (Utiliza o disco);
- Biblioteca Compartilhada (Economia de Memória);
- SWAP de Processos (Utiliza o disco);
- Segmentação Sob Demanda (Utiliza o disco);
- Paginação Sob Demanda (Utiliza o disco).

13) OVERLAY

A ideia do OVERLAY é manter na memória apenas as instruções e dados que são necessários em determinado momento. Quando outras instruções são necessárias, elas são carregadas no espaço que foi anteriormente ocupado por instruções que não são mais necessárias. Algoritmos especiais de relocação e ligação são necessários para construir os OVERLAYS.

O OVERLAY não precisa de suporte especial do SO o único trabalho do SO é carregar o módulo necessário na memória. Também não precisa de nenhum suporte especial de hardware exigindo mais do programa. Logo, o grande complicador é que quem implementa a lógica para carregar o código não é o Sistema Operacional, mas sim o próprio programador que tinha que se preocupar em chamar uma sub-rotina para carregar a parte necessária antes dela ser utilizada senão daria erro para o usuário.

O OVERLAY utiliza dois conjuntos de rotina, as que são usadas com frequência, e outras que são usadas com pouca frequência. As rotinas que são pouco utilizadas compartilham um pedaço da área de código alternadamente. O processo é o responsável pela troca de rotinas.



No exemplo ao lado a memória é de 64 Kb.

O processo tem 90 Kb.

Logo, para executar esse processo nas maquinas com este tipo de memória só era possível aplicando o OVERLAY. Conforme mostra a Figura ao lado.

Explicação do Funcionamento do OVERLAY

Temos um processo muito grande que não cabe na memória então o programa é dividido em quatro módulos: A, B, C, D cada um com 20 Kb. Somente dois módulos cabem na memoria os outros dois módulos ficam no disco e só vão para memoria quando forem solicitados. A rotina responsável por chamar estes módulos tem que ficar na memoria o tempo todo. Essa rotina se chama carrega_parte e fica no módulo A que é sempre fixo na memoria.

Então se o módulo A precisar de uma rotina que está no módulo C. O módulo A chama a rotina carrega_parte que carrega o módulo C para a memória e depois executa a rotina do modulo C. Então ao invés desse programa ocupar 90 Kb na memória ocupará apenas 50Kb. Com isso, reduz-se o espaço a ser utilizado por um processo.

Um exemplo de aplicação da época que utilizava OVERLAY é um editor de texto sofisticado capaz de imprimir em negrito e itálico.

Vantagens do OVERLAY

- Conseguir executar um programa grande em uma memoria muito pequena;
- Economizar memoria deixando nela somente o código que esta sendo utilizado;

- Não apresentar erro para o usuário devido a falta de memoria.

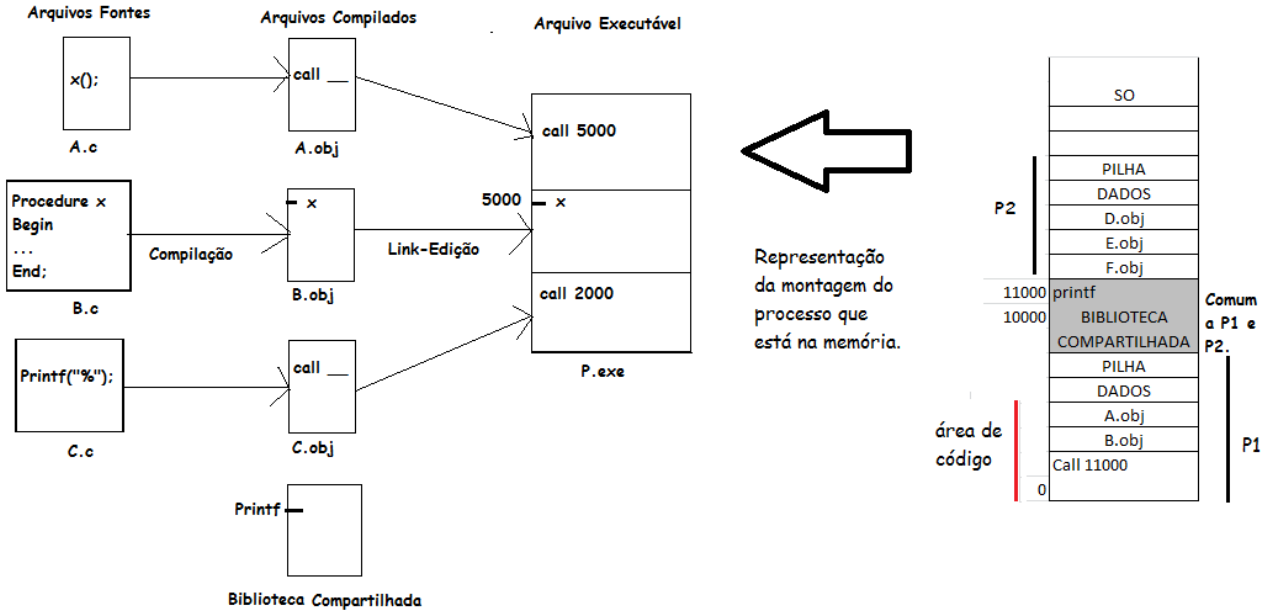
Desvantagens do OVERLAY

- Difícil de ser implementado;
- Não ter suporte do SO e nem do HW.
-

14) BIBLIOTECA COMPARTILHADA

Neste mecanismo uma parte comum a vários processos é carregada na memória uma única vez e é acessado por estes processos, ou seja, a Biblioteca Compartilhada é comum aos processos. Dessa forma não se tem código duplicado economizando memória. O SO coloca a Biblioteca Compartilhada na memória e só tira ela de lá quando nenhum processo tiver utilizando-a. O Windows chama a biblioteca compartilhada de Dynamic Link Library (DLL).

A Biblioteca Compartilhada não resolve o problema ela evita que o problema ocorra, pois diminui a memória necessária aos processos e consequentemente evita que a falta de memória ocorra.



Explicação do Funcionamento da Biblioteca Compartilhada

Temos o arquivo fonte 1 que passa pela fase de compilação e gera o arquivo objeto 1 e o mesmo acontece com o arquivo fonte 2 que também passa pela fase compilação e gera o arquivo objeto 2. Na fase da link edição esses dois arquivos são unidos para gerar o arquivo executável. E nesta fase da link edição o arquivo objeto da biblioteca padrão (rotinas disponibilizadas pela linguagem (C, Pascal)) também é adicionado, e um pedaço deste arquivo fica adicionado no executável também. Mas, no mecanismo de biblioteca compartilhada o arquivo executável não incorpora o código da biblioteca padrão, não replicando este código no executável.

O sistema operacional carrega uma única vez esta biblioteca padrão na memória, disponibilizando-a para que os processos possam encontrar a rotina solicitada. Por exemplo, quando o primeiro processo que usa a biblioteca padrão é carregado, o sistema operacional carrega o processo e a biblioteca padrão, neste nosso exemplo, P1 está sendo iniciado e precisa da biblioteca padrão então o sistema operacional carrega na memória P1 e a biblioteca padrão e quando um segundo processo, P2, for carregado na memória e solicitar uma rotina da biblioteca padrão esta já estará carregada na memória e será compartilhada pelos processos P1 e P2.

Com isso, o código da biblioteca padrão não é replicado em todos os processos que usam a biblioteca padrão. Este mecanismo é o que chamamos de biblioteca compartilhada, é um mecanismo que usa um link dinâmico e não mais estático como antigamente que acoplava uma parte da biblioteca padrão no executável.

Vantagens da Biblioteca Compartilhada

- Economizar memoria não duplicando código na memoria dos processos.
-
-

Desvantagens da Biblioteca Compartilhada

- Alguns endereços serão preenchidos pelo SO somente no momento da carga.
-
-

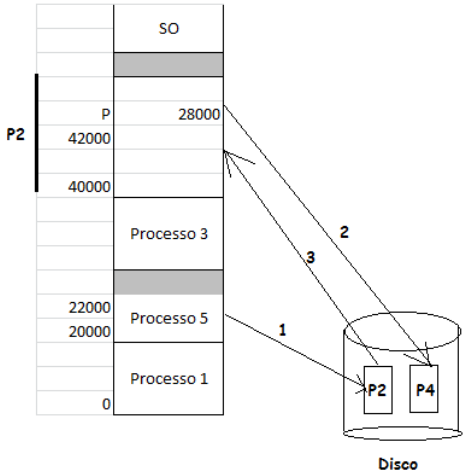
Observação

- Na biblioteca compartilhada o arquivo executável tem lacuna de endereços ainda. A link-edição não resolveu todos os problemas de endereços. Existem lacunas a serem preenchidas que são preenchidas pelo SO no momento da carga.

15) SWAP DE PROCESSOS

É quando um processo é removido temporariamente da memória e armazenado no disco e, em seguida, retornado a memória para continuar sua execução. O processo que vai para o disco é o que está no estado de bloqueado há mais tempo.

EXPLICAÇÃO DO FUNCIONAMENTO DO SWAP DE PROCESSOS



A memoria física está cheia e para o processo 5 entrar em execução é preciso selecionar um processo da fila de bloqueado para ser salvo em disco. O escolhido foi o processo 2. Conforme mostra a figura ao lado. Os números correspondem as seguintes ações do SWAP de Processo.

1) O Processo 2 foi levado para o disco através do SWAP de Processo e no seu lugar foi colocado o processo 5.

2) O Processo 4 foi levado para o disco pelo SWAP de processo, pois o Processo 2 precisa continuar sua execução e o Processo 4 não está executando (está bloqueado).

3) O Processo 2 é retornado para memoria através do SWAP, mas não volta para o endereçamento de onde foi retirado. Isto só é possível com Registrador de Base.

Uma coisa importante é a questão dos endereços. O processo 2 dois originalmente estava na memória no lugar do processo 5. Ele foi para o disco e precisou voltar. O SO trouxe de volta, mas não no mesmo endereço. O SWAP de processo tem um problema que é o mesmo processo ter que ficar em lugares diferentes na memória.

Nem sempre é possível durante a execução do processo colocar ele em outro lugar da memória. Esse é o mesmo problema que acontece na compactação. O processo 2 estava no endereço a partir do 20000 agora está a partir de 40000. Conforme mostra a Figura acima. Então o resultado final do SWAP na memória é deslocar o processo 2 na memória, mas nem sempre isso é possível por causa dos endereços que o processo utiliza.

Se o processo 2 usar endereço absoluto a partir do zero o SWAP de processos não poderá ser utilizado com este tipo de endereçamento, pois o SO não consegue corrigir o endereço da variável ponteiro. O problema é igual da Compactação. Então nesse caso o SO não permite o SWAP de Processo.

O SO permite que o SWAP de Processo aconteça com o Registrador de Base. Durante a execução do processo o SO coloca os valores do registrador de base na base do processo. Então originalmente o processo 2 está no endereço 20000. Na hora de executar a instrução o que o Hardware vai fazer? Ele vai somar o endereço contido na instrução com o valor contido no registrador de base, 20000+2000=22000. Vai chamar a subrotina no endereço 22000. Isto vale tanto para endereços de chamadas a sub-rotinas quanto para endereços de variáveis ponteiros.

O SO é responsável por mudar o valor do Registrado de base de acordo com o processo. Ele fica com todo o trabalho. Se tentar executar um processo maior que tamanho da memória da máquina, o sistema operacional tira um processo da memória e o coloca em disco para que outro processo solicitado pelo usuário seja executado.

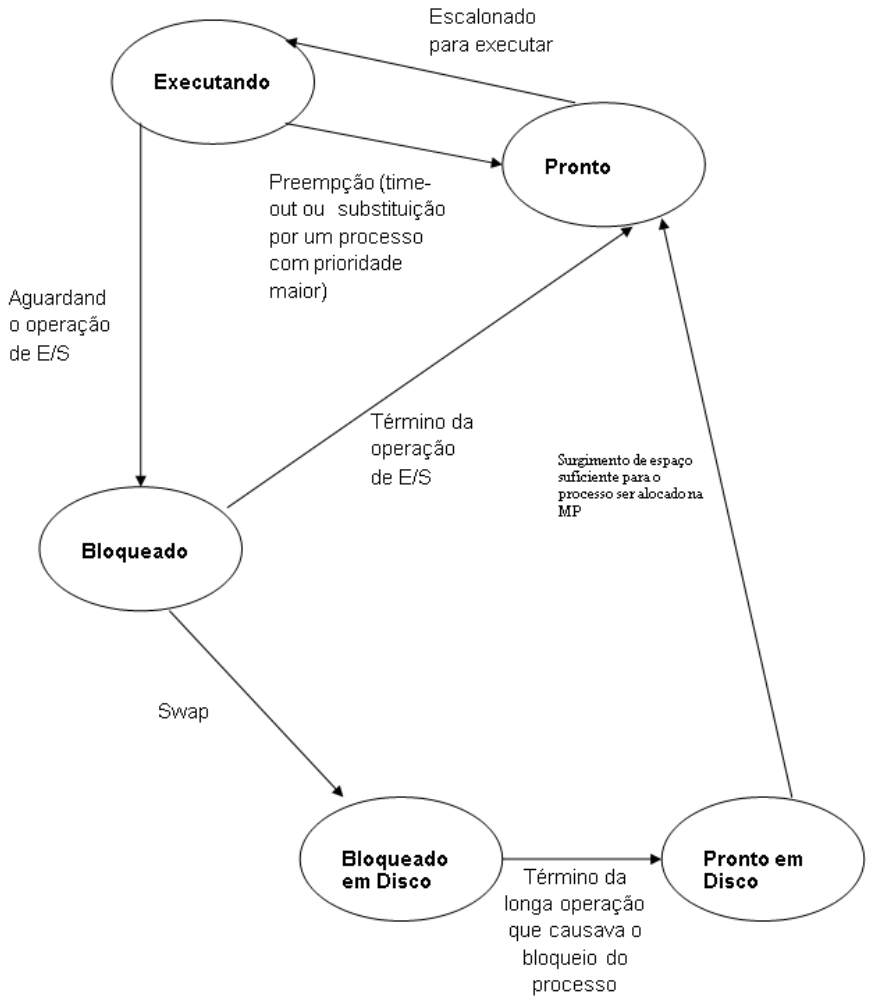
PROBLEMA DO SWAP DE PROCESSOS

O grande problema desta solução é qual o processo o sistema operacional deve escolher para retirar da memória. O importante é escolher um processo que não traga nenhum prejuízo para o usuário.

O Unix utiliza essa solução, que necessita de pouco suporte do HW. Basicamente o HW faz o carregamento dos processos em diferentes áreas da memória. O swap utiliza alguns estados a mais do que o diagrama de processos comum. Isto não tem no Windows porque no Windows não tem estado de processos.

Um processo executando não seria o escolhido pelo sistema operacional. Um processo pronto estará executando em um curto espaço de tempo, então este processo não seria o mais indicado para que o sistema operacional retire-o da memória e o coloque no disco.

Assim, o mais indicado para que o sistema operacional utilize como critério para salvar o processo em disco os processos que estão no estado de bloqueados e com o status de bloqueio longo, pois estão aguardando uma operação (leitura em disco, recepção de um pacote pela rede, resposta de um usuário etc.). Desta forma, processos bloqueados por algum tempo são os mais indicados para sofrem SWAP para disco. Quando isto acontece teremos dois novos estados para o processo que seria o estado **bloqueado em disco** e **pronto em disco.** A seguir a Figura mostra os estados do processo com SWAP de Processos.



Estados do Processo no UNIX com SWAP de Processos

Executando → Indica o processo que está na CPU sendo executado.

Pronto → Indica o processo que não está executando, mas está esperando sua vez para executar.

Bloqueado → Indica o processo que está esperando que alguma coisa acontecer, por exemplo, término de operação de I/O para que possa voltar a executar.

Bloqueado em Disco → Indica o processo que estava bloqueado na memória, mas foi salvo em disco pelo swap de processo.

Pronto em Disco → Indica aconteceu o que o processo estava esperando que acontecesse. Ele estava na fila de bloqueado em disco e agora está aguardando ir para fila de prontos na memória para em seguida continuar sua execução.

TRANSIÇÃO ENTRE OS ESTADOS

1) Um processo passa do estado **pronto** para **executando** quando ele é escalonado;

2) Um processo passa do estado **bloqueado** para **pronto** quando a causa pela qual fez o processo ficar bloqueado foi concluída. Exemplo: um clique do mouse, quando o usuário clica o processo passa para o estado de pronto;

3) Um processo passa do estado **executando** para **bloqueado** quando o mesmo solicita alguma coisa ao sistema operacional, e este fica aguardando uma resposta desta solicitação. Exemplo: uma solicitação de leitura em disco;

4) Um processo passa do estado **executando** para **pronto** quando:

a) em um sistema preemptivo: quando ocorre o fim do timelive do processo que está em execução ou a chegada de um processo com mais prioridade. Na preempção o sistema operacional tira o processo de execução por dois motivos descritos acima, sem que o processo tenha pedido nada ao sistema operacional, o processo sai involuntariamente;

b) em um sistema operacional não preemptivo: existe uma chamada que o processo (principalmente, os processos que usam muito a CPU – CPU bound) sede o lugar para outro processo. Não é preempção, pois o processo sede o seu lugar na CPU voluntariamente. Este tipo de sistema operacional é chamado de sistema operacional cooperativo.

5) Um processo passa do estado **bloqueado** em memória para **bloqueado em disco** quando um outro processo é solicitado pelo usuário e não tem espaço em memória para carregar este processo então o sistema operacional faz um swap para disco de um processo bloqueado à algum tempo na memória, e este fica bloqueado em disco;

6) Um processo passa do estado **bloqueado em disco** para **pronto em disco** quando a operação que este processo estava aguardando termina então vai para fila de prontos em disco. Ou seja, quando a causa do que o colocou em estado bloqueado não existe mais.

7) Um processo passa do estado **pronto em disco** para **pronto** em memória quando libera espaço em memória. Isto pode ocorre quando um processo libera a memória por ir para o estado de executando ou o sistema operacional pode escolher outro processo bloqueado para fazer o swap e liberar espaço para este processo pronto em disco.

Vantagens do SWAP de Processos

- Poder executar mais processos do que cabe na memória;
- O programa (programador) não precisa fazer nada. O SO é responsável por tudo.
-

Desvantagens do SWAP de Processos

- Demora em executar os processos devido a troca do processo da memoria para o disco ou vice versa;
- O processo retirado da memoria não é colocado no mesmo espaço de endereçamento que foi retirado
-

Observação

- O Swap de Processos é diferente do Overlay, pois nele o SO faz tudo enquanto que no Overlay é o programador que implementa a lógica. Além disso, o Overlay não traz todo o código que está em disco (arquivo executável) para memória quando o processo começa. No swap de processos a memória fica cheia e o SO grava todo o processo em disco e quem faz isto é o SO. No overlay quem decide o que vai para memória é o programador.
- Às vezes para colocar um processo em execução é preciso retirar mais de um processo da memoria e salvar em disco.
- Se um processo que foi salvo em disco para liberar espaço na memoria não for mais utilizado quando o computador for desligado ele será apagado do disco. Este cenário é muito raro, mas pode acontecer.

16) SEGMENTAÇÃO

A ideia básica da Segmentação é tornar a divisão do processo em áreas uma coisa conhecida pelo hardware. Para que isto aconteça cada área do processo é igual a um Segmento (numero de segmento). Conforme mostra a tabela abaixo:

ÁREA	SEGMENTO
Código	0
Dados (variáveis globais e dinâmicas)	1
Pilha (variáveis locais)	2
Biblioteca Compartilhada	3

Esses segmentos são usados pelas instruções de linguagem de máquina. Por exemplo, vou chamar uma subrotina da biblioteca: CALL 3: 1000. Onde 3 representa o Nº do segmento e 1000 o Deslocamento a partir do início do segmento.

Normalmente um CALL é um endereço a partir do zero ou a partir do registrador de base. No caso acima, tem dois campos. Esses campos são chamados de endereço bidimensional (Número do segmento e Deslocamento dentro do segmento). O primeiro campo é justamente o número do segmento. Que no exemplo acima se referente à biblioteca que pode ter várias sub-rotinas. O segundo campo é o Deslocamento (1000) a partir do inicio do segmento que identifica a subrotina que esta sendo chamada.

Para que a segmentação ocorre é preciso que o hardware seja capaz de implementar a segmentação. A Intel, por exemplo, permite. O componente de hardware (HW) responsável por controlar a memória é a Memory Mangement Unit (MMU) – Unidade de Gerenciamento de Memória.

O Deslocamento deve ser menor do que o Tamanho do Segmento. Se o Deslocamento for maior, é gerada uma interrupção de HW, e o processo pode ser abortado. Isso aumenta a proteção, pois evita que um processo mexa numa área que não lhe pertença.

Existe no HW uma Tabela de Segmentos conforme mostrada abaixo que define a localização e o tamanho do segmento. Ela tem no mínimo duas informações: Início do Segmento (na memória física) e o Tamanho do Segmento. A tabela possui dois bits de controle: Executável e Alterável. O primeiro diz se determinado segmento pode ser executado e segundo diz se o segmento pode ser alterado. O segmento de código não pode ser mudado em hipótese alguma durante a permanência na memória.

ÁREA	SEGMENTO
Código	0
Dados (variáveis globais e dinâmicas)	1
Pilha (variáveis locais)	2
Biblioteca Compartilhada	3

Tabela de Segmentos

	Inicio do Segmento	Tamanho do Segmento	Executável	Alterável
0	10000	10000	1	0
1	30000	10000	0	1
2	50000	10000	0	1
3	45000	5000	1	0

A tabela de segmentos é preenchida pelo sistema operacional e se refere ao processo em execução. Ao se mudar o processo ativo, mudam os valores da tabela de segmentos, pois mudam os segmentos e suas respectivas localizações.

Quando o processo tenta executar código em área de dados, ou alterar sua área de código é gerada uma interrupção, o processo é abortado e é gerada uma mensagem de **Segment Fault**.

Esta tabela vai ser consultada pelo HW na hora da execução do código do processo. Por exemplo, executar a instrução CALL 3:1000. Onde 3 é o nº do segmento e 1000 é o deslocamento do segmento. Várias coisas acontecem na hora da execução dessa instrução:

1ª) Verificar se o segmento é um segmento válido, pois pode não ser. A MMU verifica se o segmento é válido. O maior segmento é visto na tabela de segmentos. Neste caso é 3. Então, verifica se 3 (nº segmento informando no CALL) é maior que o máximo nº de segmento da tabela. Se for maior tem um problema. Por exemplo, se fosse CALL 5:1000. 5 > 3 . 3 é o máximo segmento da tabela de segmento. Não existe o segmento 5 no processo. Neste caso, a MMU (HW) gera uma interrupção. A rotina do SO que trata este tipo de interrupção tipicamente aborta o processo. Esta é a primeira coisa que o HW implementa para garantir a proteção de memória.

2ª) Se não gerou a interrupção. O número de segmento é usado para identificar qual o registro de segmento que será utilizado para continuar executando esta instrução. É feita uma segunda verificação. Se o valor 1000 (da instrução CALL 3:1000) é maior ou menor que Tamanho do Segmento equivalente ao Maximo Número de Segmento da Tabela de Segmentos do Processo 1 = 1000 > 5000? Não. Ok! Mas, se tivéssemos, por exemplo, 21000>5000? Sim. Isto é um problema e a MMU gera uma interrupção. Tipicamente a rotina do SO que trata este tipo de interrupção aborta o processo.

3ª) Como a segunda validação está ok! Então 1000 é somado com o valor do início do segmento, por exemplo, 1000 + 45000 = 46000 é o endereço de memória utilizada. O endereço 46000 guarda o código da subrotina a ser executada pela CPU depois que o CALL 3:1000 é executado.

Na segmentação pode se fazer a Compactação sem problemas, ou seja, mesmo após a memoria ser compactada e processo ter mudado seu espaço de endereçamento a instrução CALL 3:1000 continuará executando normalmente. O que será modificado é o campo Inicio do Segmento na Tabela de Segmentos do Processo.

Então a segmentação faz com que o HW MMU tenha um grande controle do que está na memória. Impedindo que um processo atinja as áreas do outro processo. Garantindo que um processo faça o uso correto de cada uma das suas áreas. A MMU sempre que vê alguma coisa errada gera uma interrupção e o SO trata essa interrupção. Tipicamente o SO aborta o processo com interrupção do tipo Falha de Segmento (Segmentation Fault) que é a interrupção de acesso errado a memória.

O HW acessa a Tabela através de um registrador que contem o endereço da Tabela de Segmentos do processo que está em execução. A tabela de segmentos é definida pelo SO, mas o programador pode definir o tamanho do segmento, ás vezes é o compilador.

O SO sempre aborta o processo quando ocorre uma interrupção de uso de memória errada, mais isso pode ser alterado pelo programador que criou o processo. Existe uma chamada que permite alterar o comportamento da rotina do SO que executa o tratamento deste tipo de interrupção. Isto dá uma chance do processo fazer alguma coisa útil antes de abortar.

Um exemplo onde isso ocorre e no Office da Microsoft que muda o comportamento normal do SO quando o programa comete uma falha que infringe a proteção da memória.

Por exemplo, ocorre um problema no seu documento word, ao invés do SO abortar o processo que executa o word pedi para executar uma rotina de tratamento do word que salva o que foi feito e recarrega o processo para memória. Mas nem sempre isto é possível, mas quando é possível é muito benéfico.

Vantagens da Segmentação

- Não tem endereços absolutos;
- Pode ser feita a compactação basta trocar o Endereço Inicial do segmento na Tabela de Segmento, pois o deslocamento e o tamanho são os mesmos;
- Proteção da memória, pois impedi que um processo utilize uma área da memória que não foi alocada a ele;
- Melhor controle pelo HW da memória;
- Não importa para o programa onde está o segmento de memória, pois, o programa só sabe que é um endereço bidimensional.
- Possibilidade da expansão da pilha.

Desvantagens da Segmentação

- Pode gerar Fragmentação Externa que é resolvida com a Compactação.
-
-

Observação

- Infelizmente a Microsoft quando criou o Windows 3.1 não deu uma tabela de segmento por processo, ela cria uma tabela de segmentos para todos os processos, isso é muito ruim, pois permite que um processo entre em um segmento de memória do outro, utiliza segmentação, mas não protege a memória.
- A Segmentação existe com a Paginação. No entanto, fica subutilizada, pois a Paginação é melhor que a Segmentação. A Intel possui Segmentação e Paginação em seus chips. Códigos antigos costumam funcionar somente com Segmentação. Códigos mais recentes que funcionam com Paginação.
- Nas Linguagens Interpretadas, por exemplo, Java (Máquina Virtual) existem Segmentos que podem ser Alterável e Executável ao mesmo tempo. Essa é uma exceção à regra de que um Segmento quando é executável não pode ser alterável e quando é alterável não pode ser executável.
- Para agilizar a execução da Linguagem Interpretada Java criou-se a Compilação Just In Time. Então o processo passou a ter mais uma área que contém o Código Compilado Just In Time. Esse é o Segmento com os bits Alterável e Executável da Tabela de Segmentos iguais a 1.

17) SEGMENTAÇÃO SOB DEMANDA

Ocorre quando um Segmento é escolhido para ir para disco, pois precisa liberar espaço na memoria para Segmentos de outro processo. Conforme mostra a Figura abaixo. A Tabela de Segmentos passa a ter um novo bit chamado Válido.

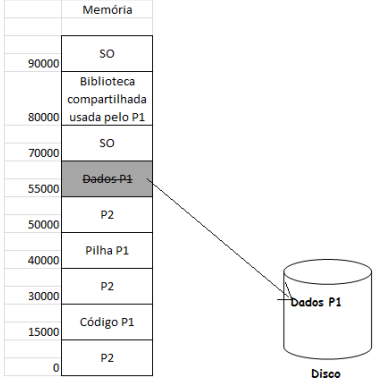


TABELA DE SEGMENTOS DO PROCESSO 1

	Início	Tamanho	Executável	Alterável	Bit válido
0	15000	15000	1	0	1
1	55000	15000	0	1	~~1~~0
2	40000	10000	0	1	1
3	80000	10000	1	0	1

Originalmente todos os segmentos estão na memoria e são validos. Quando a memoria fica cheia (ocorre falta de memória) o SO escolhe um Segmento para salvar em disco que será o Segmento Vitima. Digamos que o segmento escolhido seja o 1 que guarda os dados do processo 1. Ele será salvo no disco. O bit Valido é inativado (= 0). A área de memoria que ele ocupava passa a ser livre. Se o processo tentar usar o Segmento inválido o HW MMU gera uma interrupção para o SO trazer o Segmento de volta para memoria, pois o processo precisa dele.

Se for preciso trazer o segmento de volta para memoria dificilmente será encontrado um segmento de tamanho exato ao que foi salvo em disco. Logo pode ser que seja preciso salvar outro(s) segmento em disco. Digamos que o SO fez a escolha de outro Segmento Vitima salvou-o em disco e trouxe de volta para a memoria o Segmento 1 e atualizou a Tabela de Segmento colocando o bit Válido = 1.

A escolha do Segmento Vitima para ser salvo em disco é difícil, pois tem que vê o tamanho da área do segmento e adivinhar qual (is) segmento(s) deve(m) ser salvo em disco. Não tem um mecanismo que garanta a escolha do Segmento Vítima. Ele é simples mais menos eficiente, pois ao salvar um segmento no disco ele logo será utilizado e terá que ser trago para memoria.

Embora resolva o problema da memoria cheia salvar um segmento em disco e torná-lo invalido. A execução futura desse processo não será muito boa porque no futuro não muito distante ele não vai achar esse segmento na memoria e precisará trazer do disco.

Esse problema de ter pouco segmento por processo gera uma futura falta de segmento muito próxima. É uma coisa muito ineficiente. Portanto, embora resolva o problema da falta de memória ele é muito lento, pois sempre precisa usar o que acabou de ser salvo em disco. Ou seja, quando usa o disco a Segmentação é muito pior do que a Paginação. Este é um dos motivos da segmentação não ser utilizada atualmente.

Vantagens da Segmentação Sob Demanda

- Possui todas as vantagens da Segmentação;
- Um processo consegue ser executado mesmo com Falta de Memória;
-

Desvantagens da Segmentação Sob Demanda

- É complicado escolher o segmento vitima, pois geralmente é preciso escolher mais de um e deve ser continuo.
- Segmentos são grandes e demora a ser salvo em disco e trazê-lo de volta para memoria.
- Cada segmento tem um tamanho diferente.

Observação

- É conhecido como primo pobre da paginação, pois apesar de salvar em disco, os segmentos são bem maiores que as páginas, logo o tempo para salvar em disco e trazer do disco será grande, bem maior que na paginação. Devido a isso é bem menos eficiente que a paginação. O Windows 3.1x e o OS2 1.x usam segmentação sobre demanda.
- A diferença entre Segmentação e Paginação é que na paginação as tem o mesmo tamanho enquanto que na segmentação cada segmento poderá ter um tamanho diferente do outro.
- Na paginação um processo tem muitas paginas e ao salvar paginas em disco a execução do processo não será necessariamente afetada, pois basta o processo não utilizar esta pagina. Se ele precisar dessa pagina ela será traga para memoria sem gerar erro para o usuário ou para o processo.
- Na segmentação podemos quebrar uma área em vários segmentos, mas isso não é comum de se ver. Às vezes um segmento de dados tem 4 ou 5 segmentos. Com esta quantidade de segmento 1 para cada área teremos muita interrupção para qualquer nova execução do processo. Se este segmento for divido em outros segmentos diminuirá a chance de usar o segmento que esta em disco, mas será 1 para 5.
- Na paginação é muito mais improvável usar 1 pagina de 100 da área de dados. A chance é menor de usar a pagina que foi salva em disco. Na segmentação é muito mais provável usar um segmento que foi salvo em disco.
- O Windows 3.0 usava segmentação, mas no Windows seguinte a Microsoft deixou de usar e passou a utilizar somente a paginação porque é muito melhor. Não se utilizou paginação antes porque o chip da Intel (em 1985)

não tinha paginação quando passou a ter paginação é que o Windows (em 1995) passou a utilizar paginação. Na pratica a segmentação não se usa mais.

18) PAGINAÇÃO

A paginação faz uma divisão da memória em unidades de tamanho igual que são as páginas. O tamanho da página é igual a 4 KB. Cada área tem um número inteiro de páginas. Conforme mostram as Figuras abaixo:

		Memória Lógica de P1			Memória Fisica			Memória Lógica de P2	
Pilha P1	12	P P1 (3)		12	P P1 (1)		12	P P2 (2)	Pilha P2
	11	P P1 (2)		11	P P2 (2)		11	P P2 (1)	
	10	P P1 (1)		10	P P1 (3)		10		
	9			9	P P2 (1)		9		
	8			8	P P1 (2)		8		
	7			7			7		
	6			6			6		
	5			5	D P2		5		
	4			4	D P1 (2)		4		
Dados P1	3	D P1 (2)	HEAP	3	C P1 (1)		3		
	2	D P1 (1)		2	D P1 (1)		2		
Código P1	1	C P1 (2)	4095	1	C P2		1	D P2	Dados P2
	0	C P1 (1)	0	0	C P1 (2)		0	C P2	Código P2

Tabela de Páginas do Processo 1

	NumPagLog	NumPagFis	Válido	Executável	Alterável	Núcleo	Alterado	Acessado
Pilha P1	12	10	1	0	1			
	11	8	1	0	1			
	10	12	1	0	1			
	9		0					
	8		0					
	7		0					
	6		0					
	5		0					
	4		0					
Dados P1	3	4	1	0	1			
	2	2	1	0	1			
Código P1	1	0	1	1	0			
	0	3	1	1	0			

Toda pagina com o bit Válido = 1 na Memória Lógica tem que existir no mundo verdadeiro. Existe uma relação das páginas do mundo ilusório para o mundo real. A diferença do mundo ilusório o mundo real é que as páginas da memoria logica estão em outra ordem na memoria física.

Na prática a construção da ilusão da memória logica é feita através da conversão do endereço logico para o endereço físico, pois o processo acredita que o endereço logico é o endereço físico.

Os processos acham que existe somente a memória ilusória, mas na verdade existe apenas a memória física. Com esta ilusão o processo acha que a memória é toda dele. Assim é garantida a proteção da memória, pois, por exemplo, o conteúdo do processo 2 não aparece na memória lógica do processo 1. A memória lógica começa no endereço zero.

Usando a memória diretamente existem muito problemas, por exemplo, tem que usar registrador de base. Colocando a ilusão da memória lógica no meio do caminho resolve se todos estes problemas. Há muito tempo atrás o SO rodava um processo por vez. A memoria real da máquina tinha um processo só. Eram muito mais simples neste tempo as coisas. A paginação traz esta simplicidade de volta.

O SO vê no arquivo executado do processo o tamanho do código, o tamanho inicial de dados, o tamanho inicial da pilha e vai escolher paginas físicas livres para guardar esse conteúdo em qualquer lugar da memória física e colocará a correspondência no campo NumPagFisica da Tabela de Páginas do Processo.

Para micro o primeiro chip da Intel que tinha este mecanismo foi o 386 e o primeiro sistema operacional que começou a usar a paginação foi o Windows 3.0.

TABELA DE PAGINAS DO PROCESSO

A Tabela de Páginas é responsável pelo mapeamento da correspondência do Numero da Pagina Logica e o Numero da Pagina Fisica. Acima temos um exemplo de Tabela de Paginas do Processo. Quem preenche a tabela de página é o Sistema Operacional quando aloca o processo. Cada processo tem sua própria Tabela de Página.

Ela fica na memória do Sistema Operacional. E o HW MMU consulta esta tabela para fazer a conversão do endereço da página lógica no endereço da página física. O numero de paginas no mundo virtual é igual o numero de entradas na tabela de paginas com o bit Válido = 1.

O número da página lógica não é um campo, mas o index da tabela de páginas. É onde se guarda o valor do número da página física.

BIT VÁLIDO

Informa se a página existe ou não. Se for 0 informa que a página lógica está vazia, logo não existe na página física. Se for 1 informa que a página lógica tem conteúdo, logo existe na página física. O bit Válido = 0 também pode indicar que a página foi escolhida como página vitima e está salva no disco. O HW não sabe se a página é uma pagina vitima somente o SO. Então no tratamento da interrupção gerada pelo HW o SO verifica se terá que abortar o processo, pois a página nunca existiu ou se terá que trazer a página do disco para memoria física.

Este bit também é utilizado na **simulação do bit Acessado** que é feita pelo o SO. Ele coloca uma informação falsa no bit Válido na Tabela de Paginas do Processo. Por exemplo, coloca Valido = 0 nas paginas logicas que estavam validas e coloca a informação correta na Tabela Auxiliar no campo Realmente Valido = 1. No campo Acessado da Tabela Auxiliar o SO coloca o valor 0.

BIT EXECUTÁVEL

Informa se a página pode ser executável ou não. Se for 0 informa que a página lógica é não executável. Se for 1 informa que a página lógica é executável.

BIT ALTERÁVEL

Informa se a página pode ser alterada ou não. Se for 0 informa que não pode ser alterada. Se for 1 informa que a pagina pode ser alterada.

Também é utilizado para **simular o bit Alterado** que é feita pelo o SO. Ele coloca uma informação falsa no bit Alterável na Tabela de Paginas do Processo. Por exemplo, coloca Alterável = 0 nas paginas logicas que eram alteraveis e coloca a informação correta na Tabela Auxiliar no campo Realmente Alterável = 1. No campo Alterado da Tabela Auxiliar o SO coloca o valor 0.

BIT NÚCLEO

Informa a quem pertence a pagina. Uma página logica pode pertencer ao processo ou ao Sistema Operacional. Se for 0 informa que a página pertence ao processo. Se for 1 informa que a página pertence ao Sistema Operacional..

Através desse bit é mapeada as paginas que podem ser executadas nos modos de operação. Se bit Núcleo = 1 só podem ser executada no Modo Privilegiado (somente o SO tem permissão). Se bit Núcleo = 0 pode ser executada em qualquer Modo de Operação sendo chamada de Modo Não Privilegiado. Nesse caso tanto o SO quanto o processo podem executar.

BIT ALTERADO (DIRTY)

Informa se a pagina logica foi alterada ou não. Se for 0 informa que a página não foi alterada. Se for 1 informa que a pagina foi alterada.

Se o bit Alterado = 0, o conteúdo da memória é igual ao conteúdo do disco, então, não precisa salvar novamente. Se o bit Alterado = 1, o conteúdo da memória é diferente do conteúdo do disco, então, é necessário salvar em disco pelo menos uma vez.

Portanto, o bit Alterado indica quando uma página foi alterada durante a execução do processo. Esse bit é zerado pelo sistema operacional quando a página é carregada para a memória. A MMU automaticamente liga o bit quando o processo realiza uma operação de escrita nessa página.

Dessa forma, o sistema operacional é capaz de determinar se essa página está alterada com relação à sua cópia em disco. A maioria das páginas de um processo (código e constantes) é apenas lida e nunca escrita. Dessa forma, graças ao bit de sujeira, na maioria das substituições de páginas haverá apenas um acesso ao disco, e não dois.

Observe que o tempo necessário para atender a uma falta de página é dado basicamente pelo tempo do acesso ao disco. A disponibilidade do bit de sujeira significa que o tempo de atendimento à falta de página pode ser reduzido para a metade na maioria dos casos.

BIT ACESSADO (OU USADO)

Informa se uma pagina logica foi acessada ou não. Se for 0 informa que não foi acessada. Se for 1 informa que foi acessada.

ANALISE DO DESPERDÍCIO DE MEMÓRIA DEVIDO A FRAGMENTAÇÃO INTERNA

Digamos que eu tenha um código que tem o tamanho de 8192 Bytes. Neste caso não sobra nada, pois ocupa duas paginas. Este é o melhor caso, pois a perda é de 0 bytes. O pior caso é quando o código ocupa um byte a mais, por exemplo, 8193 Bytes. Neste caso, a perda é de 4095 bytes e a perda média é de (0 + 4095)/2 = 2047,5 bytes.

Melhor caso Fragmento Interno = 0
Pior caso Fragmento Interno > 0
Perda Média = Melhor caso + Pior caso / 2

O problema da Fragmentação Interna na Paginação não tem solução, mas a Perda Média gerada pela Fragmentação Interna não é uma perda considerável.

CONVERSÃO DO ENDEREÇO LÓGICO EM ENDEREÇO FÍSICO

O processo vai usar o endereço da memória lógica, pois ele acha que a memória lógica é a memória verdadeira. Por exemplo, alterar o conteúdo do endereço 8200: Mov [8200] AH (Este será o Exemplo 1).

O endereço 8200 é um endereço lógico. Para que o processo acesse realmente o conteúdo desse endereço na memoria física devemos fazer a conversão do Endereço Lógico para o Endereço Físico. Esta conversão é feito através do seguintes passos:

1) Quebrar o endereço lógico em duas partes: Número da Página Lógica e Deslocamento na Página;

2) A segunda coisa a ser feita é a conversão do Número da página lógica para o Número da página Física;

3) O terceiro passo é a união do Número da página Física com o Deslocamento.

QUEBRA DO ENDEREÇO LÓGICO EM DUAS PARTES

NumPagLogica = Int (EndLogico / TamPagina)

Deslocamento = EndLogico % TamPagina

Onde % é o resto da divisão inteira

Logo, cálculo os valores com o exemplo dado acima (Mov [8200] AH), temos: NumPagLogica = Int(8200/4096) = 2 e Deslocamento = 8200%(4096*2) = 8.

CONVERSÃO NumPagLogica → NumPagFisica

Através da Tabela de Páginas do Processo o HW MMU consegue descobrir a correspondência do Numero da Pagina Logica e o Numero da Pagina Fisica.

Tabela de Páginas do Processo 1

Num_pag_logica	Num_pag_Fisica	Válido
12	11	1
11	6	1
10	8	1
9	-	0
8	-	0
7	-	0
6	-	0
5	-	0
4	-	0
3	1	1
2	0	1
1	3	1
0	2	1

No exemplo em questão a Tabela de Páginas do Processo é mostrada na Figura ao lado. Podemos observar que o Numero da Página Lógica do endereço logico 8200 é 2.

Estando no NumPagLog = 2 o HW MMU verifica se a página logica 2 é um pagina válida, ou seja, se o bit de controle Válido = 1. Nesse caso, vemos que sim. Então ele prossegue com a conversão e verificar qual é o NumPagFisica que é 0.

Então: NumPagLogica → NumPagFisica = 2 → 0

Até aqui descobrimos:

	NumPagLog	Deslocamento
EndLogico	2	8
	NumPagFis	Deslocamento
EndFisico	0	8

O TERCEIRO PASSO É A UNIÃO DO NÚMERO DA PÁGINA FÍSICA COM O DESLOCAMENTO

EndFisico = (Num_pag_fisica* Tam_pagina) + Deslocamento

EndFisico = (0*4096) + 8 = 8

CONCLUSÃO EXEMPLO 1

Nesse exemplo, quando o processo tenta usar o endereço lógico 8200 o endereço físico na memória que vai ser executada de fato é 8. O processo sempre vai utilizar o endereço lógico, o endereço que ele acha ser verdadeiro. E o HW MMU faz a conversão do endereço lógico no endereço físico.

Lembro que o computador trabalha com contas binárias. Logo, as contas são diferentes da apresentada acima. O tamanho da página é sempre potencia de 2, por exemplo, 4096 = 2^{12}.

Exemplo 2

O Exemplo 1 é um caso que a conversão dá certo. Mas existem casos que a conversão não dá certo.

Por exemplo, MOV [22000], BH

NumPagLogica = Int(EndLogico/ TamPagina) = Int(22000/4096) = 5

Logo, no primeiro passo da Conversão do Endereço Logico para o Endereço Fisico ao tentar converter o NumPagLogica para NumPagFisica o HW MMU verifica que na Tabela de páginas do processo 1 o bit Válido da página lógica 5 é 0. Então o HW MMU gera uma interrupção, pois não corresponde a uma página verdadeira (não existe esta página na memoria fisica). Uma rotina do SO vai tratar esta interrupção.

CONCLUSÃO EXEMPLO 2

A conversão do Endereço Logico para Endereço Fisica não é realizada, pois não tem nenhuma logica converter o endereço logico de uma pagina que não existe na memoria verdadeira. Nesse caso a rotina do SO que trata este tipo de interrupção aborta o processo.

Exemplo 3

Por exemplo, MOV [4300], BH. Tem que fazer a conversão para o endereço físico.

NumPagLogica = Int (EndLogico/Tam_pag) = Int(4300/4096) = 1

O que a instrução MOV [4300], BH tem que fazer? Pegar o valor contido no BH e copiar para o endereço 4300. Logo, estou querendo alterar o valor contido no endereço 4300 que é uma página lógica válida, mas a pagina logica 1 não permite a alteração do conteúdo. Conforme mostra a Figura abaixo.

Tabela de Páginas do Processo 1

Num_pag_logica	Num_pag_Fisica	Válido	Alterável	Executável
12	11	1	1	0
11	6	1	1	0
10	8	1	1	0
9	-	0		
8	-	0		
7	-	0		
6	-	0		
5	-	0		
4	-	0		
3	1	1	1	0
2	0	1	1	0
1	3	1	0	1
0	2	1	0	1

CONCLUSÃO EXEMPLO 3

Esta pagina contém o código do processo que não pode ser alterado somente executado. Então dá erro, pois o HW MMU gera uma interrupção informando que a instrução está tentando alterar o conteúdo da pagina lógica 1, mas na tabela de página do processo a página lógica 1 tem o bit alterável igual a 0, ou seja, não tem permissão para ser alterada.

Logo, a rotina do SO que trata este tipo de interrupção aborta o processo.

ANALISE DO DESEMPENHO NA CONVERSÃO DO ENDEREÇO LOGICO PARA O FÍSICO

A primeira conversão do Endereço Lógico no endereço físico gasta um tempo que a principio é bem elevado, pois precisa consultar a tabela de paginas que está na memória.

A Tabela de Paginas fica na memoria e não em registrador porque é tem milhões de registros, sendo assim, muito grande. Tem um registrador que aponta para o endereço da Tabela de Páginas do Processo em execução que o HW MMU tem que usar para fazer a conversão do endereço lógico para o endereço físico.

Este registrador sempre aponta para a Tabela de Paginas do Processo que está em execução. O SO que muda o valor desse registrador. A Tabela do Processo fica na memória durante a vida do processo só apaga quando ele morre. Se o processo for bloqueado, mas depois ele vai voltar a executar. A tabela fica na memória.

Digamos que numa certa máquina que tem este tipo de memória o:
Tempo de Acesso a Memória Física = 100 ns = 100*10-9 s = 10-7 s

Então quando uma máquina sem paginação faz acesso à memória o tempo gasto é de 100 ns.

Quando tenho uma máquina com paginação e que tem uma instrução de acesso a memória, por exemplo, Mov [8200], AH. Primeiro fazemos a conversão de endereço lógico para endereço físico.

Tempo de Acesso a Memória = Tempo de Conversão EndLog em EndFis + Tempo de Acesso a Memória Física

Então pela conta percebe-se que uma máquina com paginação vai ser sempre mais lenta do que um máquina sem paginação, pois a máquina com paginação tem o tempo de conversão que a maquina sem paginação não tem.

No exemplo atual a conversão EndLogico em EndFisico acessa a tabela de páginas que está na memória física. Sendo assim, temos:

Mov[8200], AH
EndLogico → EndFisico
8200 → 8

Tempo de conversão = Tempo de Acesso a Memória Física (Acesso a Tabela de Página)

Tempo de Acesso a Memória = Tempo de Conversão EndLog em EndFis + Tempo de Acesso a Memória Física

Tempo de acesso a Memória = 100 ns + 100 ns = 200 ns

Tempo de Conversão Tempo para acessar o conteúdo propriamente dito

Este é o grande problema da Paginação a Perda de velocidade por acessar a memória, pois qualquer acesso à memória vai ser duas vezes mais lento ao que seria o acesso em uma máquina sem paginação conforme mostra o cálculo acima. No entanto, não é assim que a paginação funciona, pois existe uma solução para este problema que é a **TLB (Translation Look Asside Buffer)**.

TLB - SOLUÇÃO PARA O PROBLEMA DO DESEMPENHO NA MÁQUINA COM PAGINAÇÃO

A Tabela de Página do Processo é grande demais e fica alocada na memória sendo consultada pelo HW MMU para descobrir o endereço real de uma variável ou instrução. Para isto há 2 acessos a memória:

1. Consulta a tabela de paginas;
2. Acesso ao endereço.

Isto torna a paginação duas vezes mais lenta que um HW sem paginação. Para otimizar este processo foi criada a TLB que é consultada pelo HW MMU antes de acessar a Tabela de Paginas. A TLB está contida no HW sendo mais rápida sua consulta. Se a conversão não estiver contida na TLB a Tabela de Paginas é acessada e as paginas de consulta passam a constar na TLB.

A TLB guarda um resumo da Tabela de Páginas. A ideia é se você estiver usando uma página cuja conversão esteja na TLB vai ser mais rápido do que na Tabela de Página do Processo. Os campos que uma TLB pode ter estão descritos abaixo:

TLB de um Processo

NumPagLog	NumPagFis	Executável	Alterável	Núcleo	Alterado	Acessado	Registro em uso
12	10	0	1				
11	8	0	1				

TLB "Ampliada" de um Processo

NumPagLog	NumPagFis	Executável	Alterável	Núcleo	Alterado	Acessado	Nº do Processo
12	10	0	1				
11	8	0	1				

A TLB não tem o bit Válido, pois somente as paginas validas são salvas nela, pois são as ultimas conversões realizadas pelo HW de um determinado processo. O campo o NumPagLogica na Tabela de Páginas do Processo é um índice na TLB é um campo.

A TLB tem um tamanho reduzido, logo contém as últimas conversões feitas. Quando a página lógica é colocada na TLB é muito provável que esta página será utilizada novamente. Os registros mais antigos são retirados da TLB. Qualquer HW que tenha paginação tem TLB porque sem TLB é muito lento. Inclusive a TLB é uma parte que complica muito a paginação. A tabela é zerada quando o processo sai de execução e outro processo entra em execução. Nesse caso, a TLB será preenchida com as conversões do processo que está em execução.

Digamos que quero fazer um novo acesso ao endereço que está na página lógica 10. Na TPP isso é muito trivial porque ela é indexada pelo número da página lógica. Então para fazer a conversão basta acessar um único local na TPP, ou seja, faz um único acesso a TPP. A TLB não é assim, logo para descobrir qual o endereço da pagina lógica 10 tem que fazer uma procura até encontrar ou não.

Temos mais de um processo na memoria. Cada processo tem sua TPP. Quando o processo está em execução. O registrador aponta para TPP do processo que está em execução. O processo 1 começou a executar o SO aponta para TPP

do P1, o HW vai consultar esta tabela mais aos poucos preenche a TLB. Depois de certo tempo o HW consegue fazer muitas conversões usando somente a TLB.

Quando o processo é bloqueado para de executar e o SO escalona outro processo e aponta para a Tabela de Paginas desse processo. A TLB guarda as informações do processo atual em execução. Então não faz sentido continuar os dados do P1 na TLB. O SO tem que limpar este conteúdo. Isto é feito através do campo **Registro em uso**. Conforme mostra a figura abaixo.

Na hora que é preciso esvaziar a TLB o SO altera o campo Registro em uso para 0 (zero). Então no momento que o processo entra em execução o SO zera este campo para dizer que as informações da linha não são validas. A primeira instrução a ser executada do processo P1 vai para TLB e o campo Registro em uso é modificado para 1. Conforme mostra a Figura abaixo.

TLB do Processo 1

Num_pag_logica	Num_pag_Fisica	Alterável	Executável	Registro em uso	
1	7	1	0	~~1~~ 1	
0	2	1	0	~~1~~ 0	Não utilizado
10	8	0	1	~~1~~ 0	

Nesse tipo de TLB toda vez que o processo entra em execução ou reinicia sua execução ele é um pouco mais lento porque as primeiras conversões que o HW tem que fazer são todas via Tabela de Paginas. Pois, a TLB desse processo está vazia.

Vantagens da TLB

- Torna a conversão do Endereço Logico em Endereço Físico rápida.
-
-

Desvantagens da TLB

- Não guarda todas as conversões do processo somente as mais recentes.
- Toda vez que um processo começa ou reinicia sua execução é mais lento, pois a TLB está vazia.
-

TLB "ALTERNATIVA" OU TLB "AMPLIADA"

Existe outro componente de HW que tem TLB "Alternativa". Esta TLB tem o campo Num_Processo. Por exemplo, a TLB estava com os dados do Processo 1 conforme mostra a Figura abaixo. Mas ele foi bloqueado e o SO escalonou o processo 2. Nesse caso não precisa limpar a TLB nem invalidar o conteúdo dos campos da TLB porque tem o campo Num_Processo que diz de que tabela que veio o dado que está na TLB. Quando o P2 inicia de fato pega uma TLB vazia. O processo 2 foi bloqueado e o SO volta a executar o P1. Caso isso ocorra é vantajoso para o P1, pois ele não pegou a TLB vazia. O P1 não volta a executar tão devagar, pois já tem dados dele na TLB.

TLB do Processo 1

Num_pag_logica	Num_pag_Fisica	Alterável	Executável	Num_Processo	
~~2~~ 1	~~0~~ 7	~~1~~ 0	~~0~~ 1	~~1~~ 2	
~~0~~ 2	~~2~~ 5	~~0~~ 1	~~1~~ 0	~~1~~ 2	Não utilizado
10	8	1	0	1	

EXPLICAÇÃO DO FUNCIONAMENTO DA TLB

Acabou de começar um processo e a primeira instrução é: MOV [8200], AH
O endereço é quebrado em duas partes e descobre que o número da página lógica é igual a 2. Então o HW verifica na TLB se existe uma página lógica 2. Como o processo começou a executar agora a TLB está vazia, ou seja, não tem registro da página lógica 2. Então o HW vai ter que consultar a Tabela de Página do Processo que está na memória e descobre que a pagina logica 2 está na Pagina física 0. Então o HW salva as informações da pagina logica 2 na TLB.

Depois tem outra instrução, por exemplo, CALL 500. Olhando a TPP sabemos que neste caso o Num_pag_logica = 0 → Corresponde a Num_pag_fisica = 2. Primeiro o HW consulta a TLB, mas ela não tem os dados referentes a página lógica 0. Então o HW consulta a Tabela de Pagina do Processo da página lógica 0 e ao mesmo tempo preenche a TLB com os dados desta página. Assim quando tiver outra instrução que por sorte use uma destas duas páginas lógicas cuja conversão está na TLB será mais rápido.

Digamos que agora temos a instrução: MOV BH, [10000]. Nesse caso temos que o Num_pag_logica = 2 → Já tem na TLB. O HW consulta a TLB. Ela tem o número da página lógica que é igual a 2. Então o HW consegue fazer a conversão sem precisar consultar a Tabela de Página do Processo. Como a TLB é tipo um registrador é muito mais rápido fazer a conversão.

Vantagens da TLB Ampliada

-
-
-

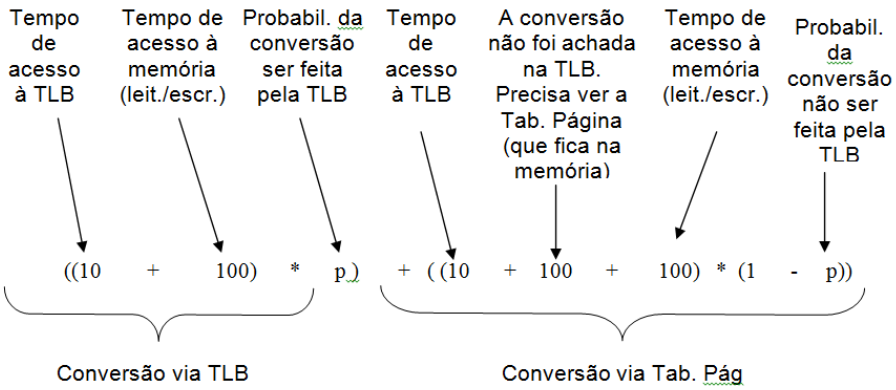
Desvantagens da TLB Ampliada

-
-
-

TLB - ANALISE DO DESEMPENHO NA CONVERSÃO DO ENDEREÇO LOGICO PARA O FÍSICO

Com o uso da TLB, o tempo reduz consideravelmente. Hoje em dia é preferível usar a paginação mesmo com perda de performance. Vamos fazer um cálculo para saber a perda de performance que a paginação pode ter utilizando a TLB:

Tempo de Acesso à Memória: 100 ns
Tempo de Acesso a TLB: 10 ns
Tempo médio de acesso à memória para instruções similares a MOV reg, [endereço]:



Efetuando os cálculos:

110 p + 210 (1 – p)

= 110 p + 210 – 210 p

= 210 – 100 p

Se em 90% dos casos a conversão estiver na TLB, o tempo médio gasto será de: 210 – 100 (0,9) = 210 – 90 = 120 ns.

Logo, a paginação traz perda de performance de 20%, pois o tempo gasto sem paginação seria de 100 ns.

Seria muito pior se não existisse a TLB, pois haveria sempre 2 acessos à memória, consumindo 200 ns.

Conversão via TLB: 10ns para consultar a TLB + 100ns para acessar a memoria depois de convertido, p é a probabilidade de acontecer a conversão via TLB.

Conversão via Tabela de Páginas: dois acessos a Tabela de Páginas (100ns + 100ns) e outro a memória para trazer o dado, também se gasta o tempo de acesso da TLB (10ns), pois o HW primeiro consulta a TLB só se não estiver nela que ele consulta a Tabela de Pagina do Processo.

No exemplo acima duas situações podem ocorrer para instruções que vai a memória:
1) O HW precisa acessar a Tabela de Páginas;
2) O HW não precisa acessar a Tabela de Páginas, pois a informação está na TLB.

Essa é uma solução razoável, porque se não houvesse nada teria uma perda de 100%, se não tivesse a TLB para fazer a conversão todo acesso a uma instrução como essa faria dois acessos a memória, então ao invés de gastar 100ns, gastará 200ns tendo uma perda de 200%. Usando a TLB a perda é de 20%. Justifica-se então que a TLB é uma solução razoável

para perda de velocidade. Uma máquina com paginação tem perda desempenho que uma maquina sem paginação não tem.

É muito importante a TLB conseguir fazer a maior parte das conversões porque só assim que não vai ter perda de desempenho.

SOLUÇÕES PARA O PROBLEMA DO TAMANHO DA TABELA DE PAGINAS DO PROCESSO

Na Intel temos 1 milhão de páginas lógicas isto não corresponde a página física sendo utilizadas pelo processo, pois temos milhares de páginas lógicas sem uso na memória lógica. Isso não gasta chip de memória, porém cria o problema do Tamanho da Tabela de Páginas do Processo. Porque se tem 1 milhão de registros na memória lógica tem que ter o mesmo valor na Tabela de Páginas. Logo, gasta se uma memória razoável somando todas as Tabelas de Páginas todos os processos que estão em execução, pois temos uma Tabela de Página por processo.

Normalmente o tamanho do espaço lógico é o tamanho máximo que a máquina permite, ou seja, o maior endereço que ela pode usar. Por exemplo, na CPU Intel o maior endereço que ela pode usar é 32 bits, ou seja, $2^{32}-1$ = 4GB. É o caso em que todos os bits são 1, já que não tem limite para o tamanho do espaço lógico, então os Sistemas Operacionais podem escolher que o espaço lógico é 4GB, geralmente é isso que acontece, por exemplo, no Windows o espaço de endereçamento virtual de um processo qualquer é de 4GB.

No interior da CPU poderá ter um outro valor de tamanho máximo que pode ser usado, o valor de 32bits pode ser 64bits, por isso, vai ser muito maior o tamanho do espaço lógico.

Número de páginas lógicas = 4 GB (Tamanho da Memoria Logica) / 4 KB (Tamanho da página lógica) = 1 Milhão de páginas. Se há 1 Milhão de páginas lógicas a tabela de página tem que ter 1 Milhão de entradas. Se cada entrada da Tabela de Páginas tem 4 bytes de tamanho, só a Tabela de Páginas ocuparia 4 * 1 M = 4 MB. Se a Tabela de Páginas tem 1 Milhão de páginas e a maioria das entradas da Tabela de Página tem o bit Válido = 0, pois não há informação válida nessas entradas então temos espaço inútil sendo gasto.

Nº páginas lógicas = Capacidade de endereçamento (Tamanho da Memoria Logica) / TamanhoPag = 4 GB/4 KB = 1 MB
Qtd_pag_logica = Qtd_registradores (Entradas) na tabela de pags = 1 MB entradas
Tam_Tab_Pags = Qtd_Entradas_Tabela * Tam_Entrada = 1 MB * 4 B = 4 MB

É vantajoso ter muitas paginas logicas sem uso entre a área de dados e a área de pilha porque isso permite crescer essas áreas sem muita preocupação em bater uma na outra. Também permite ter outros tipos de áreas no meio do caminho.

No entanto, se 1 processo gasta 4 MB para guardar a Tabela de Páginas na memória. 20 processos vão gastar 100 MB. Que é um gasto considerável de memória. Nenhum fabricante trabalha com este tipo de TPP. Todo tem alguma técnica para reduzir o tamanho da Tabela de Paginas.

Temos 3 mecanismo para resolver(ou evitar) o problema da Tabela de Páginas muito grande:

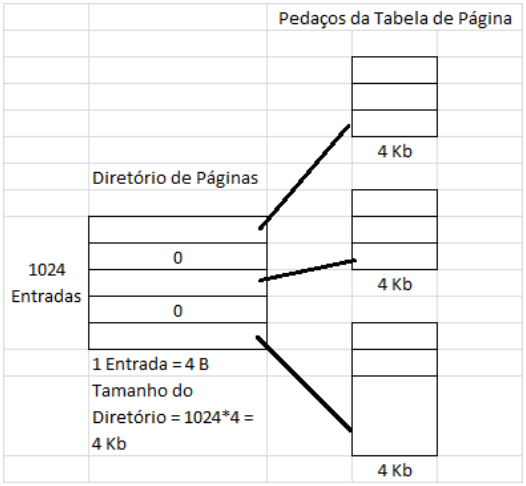
1) Tabela de Páginas em Níveis;
2) Tabela de Páginas Invertidas;
3) Geração de Interrupção quando a conversão não está na TLB.

TABELA DE PÁGINAS EM NÍVEIS (DIRETÓRIO DE PÁGINA)

A Tabela de Páginas em Níveis é a Tabela de Paginas do Processo quebrada em pedaços de tamanhos iguais na memoria. Muito desses pedaços não tem nenhuma pagina logica valida. A ideia é não gastar memoria com estes pedaços.O Diretório de páginas contém os endereços das páginas lógicas válidas da Tabela de Páginas do Processo (que não é mais inteira nessa solução inteira, mas sim em pedaços).

Esta solução resolve o problema do Tamanho da Tabela de Páginas, mas cria outro problema que é o aumento no tempo gasto para fazer a conversão do endereço da página lógica no endereço da página física. Pois, para saber em qual a pagina física está uma pagina logica primeiro o HW tem que ir ao Diretório de Páginas e vê qual o endereço do pedaço da entrada na Tabela de Página para só então acessar o pedaço da Tabela de Pagina.

Tem uma logica para o SO saber qual entrada será consultada. O 1º passo para conversão da página logica no endereço físico é quebrar o endereço em duas partes. Se a pagina for de 4 KB é 2^{12} então são 12 bits para o deslocamento. Os 20 bits restantes que ficam no lado esquerdo correspondem o numero da pagina logica.



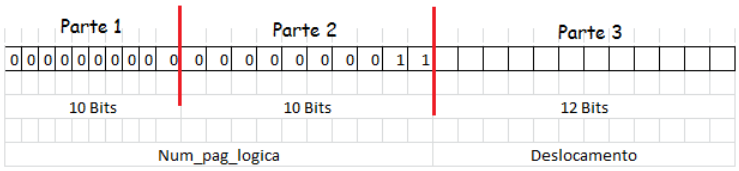
O exemplo ao lado é de uma Tabela de Páginas em dois níveis.

Tamanho do Pedaço da Tabela de Páginas = Qtd_Entradas_Pedaço * Tam_Entrada = 1024*4 = 4 KB

No exemplo da Figura ao lado foram gastos com a Tabela de Páginas 12 KB (3 pedaços) + 4 KB (Diretório). Totalizando 16 KB. Um processo maior vai ter mais paginas validas e vai dar um valor maior de memória gasto, mas mesmo assim o gasto é menor do que a Tabela de Páginas.

O registrador aponta o Diretório de Página que diz qual pedaço da Tabela de Página será utilizada. Quando o SO escalona outro processo o registrador aponta para outro Diretório.

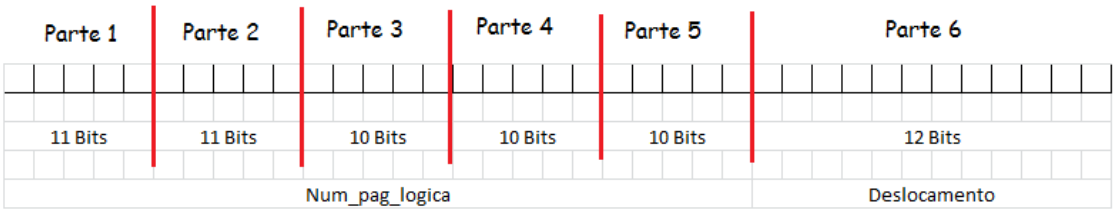
Na Tabela de dois níveis o endereço da página logica é quebrado em dois e pedaços. Conforme mostra a imagem a seguir:



A Parte 1 da Figura ao lado identifica uma entrada no diretório. A parte 2 identifica uma entrada no pedaço da Tabela de Páginas. Cada pedaço vai de 0 a 1023. E a parte 3 identifica o byte da página na memória lógica do processo. O deslocamento significa o numero do byte dentro da página logica que tem 4 KB.

Essa solução não é boa para máquinas de 64 bits (Capacidade de endereçamento 2^{64}-1), pois este tamanho é gigantesco. Sabemos que 26 bits é igual a 2^{26} = 64 MB Entradas. Logo, o diretório teria este tamanho de entradas. Sendo 4 Bytes por entrada temos 256 MB. Isto só para um processo. Portanto a lógica aplicada para 32 bits não pode se manter para 64 bits porque o Diretório de Paginas e os pedaços da Tabela de Páginas ficam gigantescos.

O Diretório de Páginas com 64 bits deve conter 5 nives. Onde o 1º nível (que seria o diretório) aponta para o 2º nível. O 2º aponta para o 3º nível e assim sucessivamente até chegar ao 6º nível. Conforme mostra a Figura temos uma entrada para tabela de pagina para cada nível.



A parte 1 da Figura acima identifica uma entrada no 1º nível. A parte 2 identifica uma entrada no 2º nível. A parte 3 identifica uma entrada no 3º nível. A parte 4 identifica uma entrada no 4º nível. A parte 5 identifica uma entrada no 5º nível. O 6º nível é referente ao pedaço da Tabela de Paginas.

Esse caso é um caso bem teórico a Intel não usa todos os 64 bits de endereço ela simplifica e usa um endereço menor. A tabela de páginas da Intel tem 5 níveis sendo 4 de diretório e 1 referente ao pedaço da tabela de paginas.

A solução funciona, mas é insuficiente na hora da conversão, pois tem que ser feito 6 acessos a memoria. Esta é a desvantagem dessa solução para 64 bits. Essa Tabela a TLB só guarda as conversões do último nível.

ANALISE NO TEMPO DE CONVERSÃO DO ENDEREÇO LOGICO NO ENDEREÇO NA TABELA EM NIVEIS

Tempo de Acesso a memoria: 1) via TLB: 10 ns 2) Via ao Diretório de Páginas e Tabela de Páginas: 100 ns

Temos 2 acessos à memória: o 1º Acesso ao Diretório de Páginas e o 2º à Tabela de Páginas. Logo, o tempo médio de acesso à memória usando Diretório de Páginas com 90% de hit na TLB:
= (10 + 100) p + (10 + 100 + 100 + 100) (1 - p) = 130 ns

Sendo via TLB: (10 + 100) p e (10 + 100 + 100 + 100) (1 - p) via Diretório de Páginas

Vantagens da Tabela de Páginas em Níveis

- Somente o Diretório de Paginas contendo pelo menos uma pagina ativa são mapeados. Como a maioria das paginas virtuais fica ociosa, o numero de paginas mapeada é relativamente pequeno.
-
-

Desvantagens da Tabela de Páginas em Níveis

- Fica mais lento, pois usa mais uma tabela. Se a pagina não estiver mapeada na TLB, são necessários 2 acessos a memória para encontrar a pagina real. No caso das máquinas com 64 bits serão necessários mais de dois acessos a memoria ficando ainda mais lento.
- A TLB é zerada a cada chamada externa para troca do espaço do processo.
-

TABELA DE PÁGINAS INVERTIDAS

Tabela de Páginas Invertidas

Num_pag_Fisica	Num_pag_logica	Processo	Está em uso
12	N	2	1
11	N	1	1
10			0
9			0
8			0
7			0
6			0
5			0
4	1	1	1
3	0	2	1
2	0	1	1
1	3	1	1
0	2	1	1

Uma tabela de pagina invertida tem uma entrada para cada pagina física. Cada entrada consiste no endereço virtual da pagina armazenada naquela posição de memória real. Assim, só existe uma Tabela de Paginas Invertida no sistema e só tem uma entrada para cada pagina de memória física.

Na Tabela de Páginas Invertida ao lado observamos que a Pagina Logica 0 pode ter dois endereços de página físicas diferentes. Logo tem que ter uma informação na Tabela Invertida que não tem na Tabela de Páginas que é o numero do processo. E o bit de páginas válidas sendo 0 para página não válidas e 1 para páginas válidas.

Esse esquema aumenta o tempo necessário para pesquisar na tabela quando ocorre uma referencia de pagina. Como a tabela de pagina invertida é classificada por endereços físicos, mas as pesquisas são feitas com endereços virtuais, a tabela inteira talvez precise ser pesquisada para encontrar uma correspondência.

Essa pesquisa pode demorar muito. Para minimizar o problema utiliza-se a Tabela de Hashing que limita a pesquisa para uma entrada, ou no máximo algumas poucas entradas na Tabela de Paginas Invertida.

Cada acesso a Tabela de Hashing acrescenta uma referencia de memória ao procedimento, por isso uma referencia de memória virtual requer pelo menos duas leituras de memória real: uma para a entrada na Tabela de Hashing, outra para a Tabela de Pagina Invertida.

Para melhorar o desempenho, utilizamos os registradores de memória associativa para manter entradas recentemente localizadas. Esses registradores são pesquisados em primeiro lugar, antes que a Tabela de Hashing seja consultada.

A Tabela de Páginas Invertida é igual à Tabela de Páginas.

EXPLICAÇÃO DO FUNCIONAMENTO DA TABELA DE PÁGINAS INVERTIDA

A instrução do processo 1→ MOV X, 1200. Esse endereço 1200 é um endereço logico que o processo 1 está utilizando e tem que ser convertido para o endereço físico. Fazendo a conversão encontramos página logica 2 e deslocamento 1. Os 12 bits de deslocamento não mudam. Mas, para fazer a conversão tem o passo 2 que é descobrir qual é a página física que corresponde a pagina logica em questão.

O dado que não tenho é justamente o endereço da página física. O dado que tenho é o endereço da pagina logica e a página física tem que ser procurada. Em qual página física esta o Num_pag_logica = 1 do processo 1? Procura na Tabela de Paginas Invertida de baixo para cima até encontrar. Nesse caso para fazer a conversão foram feitos 5 acessos a Tabela Invertida para encontrar o End_Fisico da página logica 1 do processo 1.

End_logico	1	6
	Num_pag_logica	Deslocamento
End_fisico	?4	6
		Deslocamento

Nesse exemplo o cenário é muito bom poderia ser pior a página poderia estar lá em cima da Tabela de Páginas Invertida. Logo, a procura é feita através da Tabela Hash, para ganhar velocidade de acesso. Por causa disso, a TLB é bem mais eficiente, ou seja, a conversão será melhor se for feita via TLB ao invés da tabela de páginas invertida.

É uma solução complicada utilizada pela IBM nas suas máquinas de 64 bits é uma solução interessante para estas máquinas de 64 bits.

Espaço gasto com a Tabela Invertida

Qtd de entrada * Tamanho entrada = 1 MB * 8 B = 8 MB
Qtd de entradas de Tabela Invertida = Qtd Páginas Físicas
Qtd Páginas Físicas = Tamanho da Memória Física / Tamanho da Página = 1 MB

Tabela Hash – Solução para o problema da procura na Tabela de Páginas Invertida

O HW que usa Tabela de Páginas Invertida precisa usar uma Tabela de Hash para aumentar a velocidade na procura de uma página lógica. A TLB precisa ser muito eficiente para compensar a lentidão da Tabela de Páginas Invertida.

Durante a execução o HW tem que fazer a conversão de página logica para pagina física. Antes de ir na Tabela de Páginas Invertida o HW aplica a Função Hash. Digamos que o resultado da função seja, por exemplo, F(1,1) = 2. Ele usa o resultado desse função para ir na Tabela Hash.

Nesse caso, vai na posição 2 e encontra a Página Física do processo que esta procurado que é 4. Para confirmar o resultado o HW vai na posição 4 da Tabela de Páginas Invertida e verifica se de fato a página que está procurando é do processo 1. Nesse exemplo, são gastos dois acessos a memoria.

Tabela de Páginas Invertidas

Num_pag_Fisica	Num_pag_logica	Processo	Está em uso	Próximo elemento da lista de colisão
13				
12	10	2	1	-1
11	10	1	1	-1
10				
9				
8				
7	2	2	1	-1
6				
5	1	2	1	-1
4	1	1	1	3
3	0	2	1	-1
2	0	1	1	-1
1	3	1	1	7
0	2	1	1	5

Digamos que agora temos o processo 2. O SO preenche a Tabela de Páginas Invertida e a Tabela Hash com os dados do processo 2. Ao aplicar a função Hash F(0,2) = (0+2) % 14 = 2 verifica que na Tabela Hash esse campo já está em uso.

Logo, tem um problema que se chama Colisão. A colisão pode ser resolvida de várias formas, mas aprenderemos a solução da criação de uma Lista Encadeada de Colisão para isto coloca-se mais campo na Tabela de Páginas Invertida que se chama Próximo Elemento da Lista de Colisão. Esse campo é um numero inteiro.

Tabela Hash

	Num_pag_Fisica
13	
12	
11	
10	
9	
8	
7	
6	
5	
4	1
3	0
2	4
1	2
0	

Então nesse exemplo na posição 2 da Tabela Hash encontra se a página lógica 4 do processo 1. Temos uma colisão dessa pagina com a pagina logica 0 do processo 2. Logo, colocado se no campo próximo elemento da lista de colisão o número da página logica que ele colidiu. Quem não colidiu não tem próximo e este campo fica sinalizado com -1.

Cálculo da Função Hash para o processo 2:

F(1,2) = 3
F(2,2) = 4 → Colisão
F(10,2) = 12

Digamos que temos a instrução do processo 2 → CALL 500. Esse endereço logico 500 tem que ser quebrado em duas partes:

End_logico	0	500
	Num_pag_logica	Deslocamento
End_fisico	?	500
		Deslocamento

Agora para descobrir o Endereço Físico temos que saber o número da página física. O HW:

(1º) Aplica-se F(0,2) = 2;

(2º) Vai à posição 2 da Tabela Hash e vê o valor que está lá, 4;

(3º) Vai à posição 4 da Tabela Invertida e pergunta ai está a página logica 0 do processo 2? Não.

(4º) Verifica o campo Próximo elemento da lista de colisão e vai à posição informada por este campo e pergunta se ai está a pagina logica 0 do processo 2? Sim. Então ele descobre o que queria que é o endereço físico da pagina logica 0 do processo 2 que é 3.

Nesse caso a quantidade de acessos à memória será menor do que a quantidade de acesso da Tabela de Páginas em Níveis. A Tabela de Páginas Invertida a pesar de ser muito mais complicada ela vai gerar tipicamente menos acesso a memória do que uma Tabela de Níveis. No exemplo, temos 3 acessos. Se tiver 2 colisões serão 4 acessos a memoria. A ideia é ter poucas colisões.

Portanto, para 64 bits é mais vantagem usar uma Tabela de Páginas Invertida do que usar uma Tabela com Níveis. Porque tem menos acesso a memoria do que tem em uma Tabela de Níveis.

A Tabela Hash é sempre associada a uma Função Hash: F(campos) → Nº Inteiro → Indexa a Tabela Hash

Nesse caso quero procurar o numero do processo e o numero da página logica. Logo, terei a seguinte Função Hash: F(Nº da Página Lógica, Nº do Processo) → Inteiro

Quando cada processo é criado o SO preenche a Tabela Invertida e também preenche a Tabela Hash. Então vamos supor que começou o processo 1. O SO preencheu a Tabela de Páginas Invertida e a Tabela Hash com os dados do processo 1. Mas, para preencher a Tabela Hash vai ter que usar a Função Hash para cada pagina logica do processo 1.

F(Nº da Página Lógica, Nº do Processo) → (Nº da Página Lógica + Nº do Processo) % Tamanho da Tabela Hash

Fazendo o cálculo da Função Hash para cada página logica do processo 1. O resulta dessa função é o numero da página física onde está à página logica do processo. Onde está a página lógica do processo 1? Na pagina física 1 (encontrada através da F(0,1)) é esse valor que é colocado na página física da TH. O campo da TH é o numero da página física.

Cálculo das Funções De Hash para o processo 1:
F(0,1) = (0+1) % 14 = 1
F(1,1) = (1+1) % 14 = 2
F(2,1) = (1+2) % 14 = 3
F(3,1) = (3+1) % 14 = 4

A Tabela Hash é única assim como a Tabela Invertida.

Vantagens da Tabela de Páginas Invertida

- Economia de espaço, pois há menos paginas reais que virtuais. Tem uma tabela para todos os processos;
- Não precisa zerar a TLB quando faz uma chamada ao sistema operacional sob a forma de uma interrupção, porque o espaço do endereçamento lógico é o mesmo.
-

Desvantagens da Tabela de Páginas Invertida

- Toda pagina real é mapeada, tendo conteúdo ou não, mapeamento muito lento, pois é necessário um acesso a memória para cada entrada, até achar a pagina virtual desejada e depende do funcionamento da TLB;
- Problemas para conseguir fazer a conversão do Endereço da Página Lógica para Endereço da Página Física, pois é necessário varrer toda a Tabela de Páginas Invertida até encontrar a Página Física que tem como referência uma determinada Página Lógica usada por um determinado processo.
-

COMPARAÇÃO DA TABELA DE PÁGINAS INVERTIDA COM A TABELA DE PÁGINAS EM NÍVEIS

Geralmente a Tabela de Páginas Invertida com a Tabela Hash gasta 2 acessos a memoria para fazer a conversão do endereço logico para o endereço físico. No caso mais demorado vai gastar mais acessos.

Na Tabela de Páginas em Níveis tem um numero fixo de acessos que é igual ao numero de níveis. Uma Tabela de Páginas em Níveis com muitos níveis o numero de acesso a memoria é muito elevado.

A Tabela de Páginas Invertida se gasta muito menos memoria do que com a Tabela de Páginas em Níveis.

Na Tabela de Páginas Invertida com a Tabela Hash é quase improvável acontecer do numero de acessos a memoria para fazer a conversão ser igual ao numero de acesso da Tabela de Páginas em Níveis. Só em casos muito raros isso pode acontecer.

Embora seja muita mais complexa a Tabela de Páginas Invertida com Tabela Hash é mais vantajoso usá-la em máquinas de 64 bits porque gera menos acesso a memoria para fazer a conversão do endereço logico para o endereço físico.

GERAÇÃO DE INTERRUPÇÃO QUANDO A CONVERSÃO NÃO ESTÁ NA TLB

Nessa solução o HW tenta fazer a conversão na TLB e se conseguir é ótimo, se a conversão não tiver na TLB então o hardware gera uma interrupção e o sistema operacional trata esta interrupção, fazendo a conversão, usando estrutura de dados feita em software e não hardware, ou seja, quando a pagina virtual não esta mapeada na TLB, o HW gera uma interrupção e o software fica responsável por encontrar a pagina real correspondente.

Essa conversão de lógico para físico é uma coisa complicada então as estruturas de controle estão amarradas ao hardware. A ideia dessa solução é deixar livre a escolha da estrutura de controle. Quem vai decidir qual o melhor mecanismo será o programador do SO. Atualmente essa solução não existe mais no mercado.

Vantagens da Geração da Interrupção

- HW é mais simples e fica livre. Mais fácil de implementar e reprogramar a solução de mapeamento via software.
-
-

Desvantagens da Geração da Interrupção

- O Sistema Operacional precisa fazer a conversão tornando-a mais lenta.
-
-

BIBLIOTECA COMPARTILHADA NA PAGINAÇÃO

O SO faz com que as páginas logicas dos processos que utilizem a mesma DLL apontem para as mesmas páginas fisicas da Biblioteca. Sendo assim é aproveita a mesma DLL para vários processos.

A paginação, assim como na segmentação, facilita o compartilhamento de procedimentos e/ou dados entre vários processos. Um exemplo é uma biblioteca compartilhada. Neste caso, os procedimentos desta biblioteca permanecem em algumas páginas físicas que podem ser utilizadas por diversos processos, sem que cada um precise possuí-la em seu espaço de endereço físico somente no espaço de endereçamento logico.

Vantagens da Biblioteca Compartilhada

-
-
-

Desvantagens da Biblioteca Compartilhada

-
-
-

SISTEMA OPERACIONAL NA PAGINAÇÃO

Na paginação o Sistema Operacional ocupa a memória lógica de duas maneiras:

1) Ter sua própria Memória Lógica; Ou
2) NA Memória Logica de todos os processos.

O SO ESTAR TER SUA PRÓPRIA MEMÓRIA LÓGICA

Com TLB "Simples"

	Memória Lógica Processo 1			Memória Física Verdadeira		Memória Lógica do Sistema Operacional
11	Pilha P1		11		11	Pilha SO
10			10	Pilha P1	10	
9			9	Pilha SO	9	
8			8		8	
7			7	Pilha P1	7	
6			6	Dados P1	6	
5			5	Dados SO	5	
4			4	Dados P1	4	
3	Dados P1		3	Código SO	3	
2			2	Código P1	2	Dados SO
1	Código P1	4095	1	Código SO	1	Código SO
0		0	0	Código P1	0	

TLB do Sistema Operacional

Num_pag_logica	Num_pag_Fisica	Executável	Alterável
1	3	1	0
2	5	0	1
11	9	0	1

A TLB é um resumo da Tabela de Paginas que guarda o numero da pagina logica. Como o SO é um processo. Ele também terá uma TLB conforme mostra a Figura acima.

Quando o processo começa a rodar a TLB não tem dado nenhum e é preenchida conforme as paginas logicas vão sendo acessadas. Por exemplo, o processo 1 mostrado na Figura acima rodou um pouco e o HW preencheu a TLB com alguma informação do processo 1: pagina logica 0 está na página física 2 (executável e não alterável). A pagina logica 2 está na pagina física 4 (não executável e alterável). E assim por diante. Assim a conversão pode ser feita via TLB que é mais rápido.

Quando o código do processo esta rodando o HW faz as conversões na tabela de paginas do processo e os dados de conversão são colocados na TLB. De tempos em tempos ocorrem interrupções. Tem interrupções que é motivada por eventos de HW e outras são chamadas que o processo faz ao SO. Então ele entra em execução não só para tratar a interrupção, mas também porque o processo o chamou.

Nesse caso em que o SO tem a memória logica só dele quando ele entrar em execução seja porque ocorreu uma interrupção, ou seja, porque um processo fez uma chamada ao SO. O SO precisa rodar o código dele e acessar a memoria logica dele. Só que a memoria logica dele tem outra memória física. A pagina logica 0 do Processo 1 está na pagina física 2 e a pagina logica 0 do SO está na pagina física 1.

Então as conversões que se encontram na TLB elas são validas enquanto o código do processo 1 estiver executando. Se entrar em execução o código do SO as conversões que se encontram na TLB não valem para ele. Logo o HW esvazia a TLB, pois não é valida para a memoria logica do SO.

Conclusão quando ocorre uma interrupção ou quando um processo chama o SO o HW precisa limpar o conteúdo da TLB. Este tipo de TLB tem que ser limpa ao trocar o processo a ser executado. E as conversões do outro processo, nesse caso, do SO são colocadas na TLB.

Isto é um problema, pois toda vez que é preciso esvaziar a TLB as conversões seguintes são feitas usando a Tabela de Paginas do processo que está em execução o que torna lenta a execução do processo, pois só depois e preenchida a TLB.

A memoria logica do SO e independente da memoria logica do Processo. Quando ocorre uma chamada ao SO ou quando ocorre uma interrupção o SO passa a executar mais devagar, pois as conversões de pagina logica para pagina física tem que ser feitas via Tabela de Páginas.

É muito mais frequente o SO entrar em execução do que outro processo. Tipicamente para outro processo entrar em execução ele é bloqueado ou acaba seu tempo-limite (quantum ou time-slice). Que é uma coisa que não ocorre muitas vezes por segundo ocorre algumas vezes. As interrupções e chamadas ao SO podem ocorrer centenas por segundo. É um numero muito maior. Por exemplo, a interrupção chamada de relógio ocorre 100 por segundo.

Portanto, é muito mais frequente o código do SO entrar em execução do que o código do processo. O processo pode fazer varias chamadas ao SO. A cada chamada ao SO tem que rodar o código do SO. A cada execução do SO o HW terá que limpar o conteúdo da TLB fazer as conversões via Tabela de Páginas e preencher a TLB. Assim, o sistema como todo fica mais lento.

Com TLB "Ampliada"

Alguns HW tem outro tipo de TLB chamada de TLB "Ampliada" ou TLB com o Nº do Processo. As conversões do SO são representadas no campo "Nº do Processo" com o bit zero. E as do Processo com o bit um.

TLB com o número do processo

Num_pag_logica	Num_pag_Fisica	Executável	Alterável	Nº do Processo	
0	2	1	0	1	
2	4	0	1	1	
10	7	0	1	1	
0	1	1	0	0	Memória Lógica
2	5	0	1	0	do SO.

Além dos campos convencionais da TLB "Simples" tem mais um o "Nº do Processo". Por exemplo, o processo 1 começou a executar e a TLB está fazia. Então o HW faz as conversões via Tabela de Páginas e preenche a TLB aos poucos com as conversões do processo 1 (Vide Figura ao lado).

Ocorre uma interrupção ou uma chamada do SO. Então tem que executar o código do Sistema Operacional. Tem que mudar a memoria logica. Só que na TLB "Ampliada" não precisa limpar seu conteúdo, pois o campo Nº do Processo diz de que processo veio a conversão. Quando ocorre uma interrupção ou é feita uma chamada ao SO.

Quando o SO entrar em execução novamente, como as conversões do SO não foram retiradas da TLB "Ampliada" algumas conversões poderão ser feitas via TLB deixando a execução do SO mais rápida, pois é muito provável que ainda tenha na TLB dados de conversão a respeito do SO.

Esta solução de ter o SO na memoria logica dele é uma solução que tem perda de desempenho caso o HW seja uma TLB "Simples", mas se o HW tiver uma TLB "Ampliada" não teremos problemas, pois possivelmente o HW encontrará conversões do SO na TLB "Ampliada" não precisando acessar a Tabela de Páginas tornando a execução mais rápida.

Nessa solução de se ter o SO na mesma Memória logica do processo. Na memoria logica do processo existem paginas validas que são do SO. A Figura 17 mostra a Tabela de Página do Processo 1 existem páginas logicas que pertencem ao SO e não ao processo.

Tabela de Páginas do Processo 1

Num_pag_logica	Num_pag_Fisica	Válido	Executável	Alterável	Núcleo
11	9	1	0	1	1
10	-	0			
9	10	1	0	1	1
8	7	1	1	0	1
7	8	1	0	1	0
6	4	1	0	1	0
5	-	0			
4	-	0			
3	-	0			
2	4	±0	0	1	0
1	5	1	1	0	0
0	1	1	1	0	0

O que impede o processo não alterar a pagina logica do SO que está na mesma memoria logica do processo é o bit de controle Núcleo mostrado na Figura ao lado.

Esse bit foi criado para que o mecanismo do SO está na mesma memoria logica do processo seja possível de ser implementado garantido a estabilidade do processo e do SO.

No campo Núcleo é preenchido pelo SO com o valor o 1 quando é uma pagina do SO (Modo Privilegiado) e com 0 quando é uma pagina do processo (Modo não Privilegiado). Estes são os Modos de operação: Privilegiado e Não Privilegiado.

Esvaziar a TLB é uma coisa que é feita no modo privilegiado. Quando estiver em execução o código do processo o HW sabe o que é do processo e o que é do SO e não permite que o processo altere o código do SO. As paginas com o bi Núcleo igual a 1 só podem ser executada no Modo Privilegiado e as paginas com o bit Núcleo igual a 0 podem ser executadas em qualquer modo de operação.

Vantagens do SO ter sua própria Memória Lógica

-
-
-

Desvantagens do SO ter sua própria Memória Lógica

-
-
-

O SO ESTAR NA MEMÓRIA LÓGICA DE TODOS OS PROCESSOS

Nessa solução o processo não ocupa todo o intervalo da Memoria Logica. A ideia do processo ter a Memória Lógica só para ele não existe. A distribuição da Memória Lógica do processo com o SO nela fica da seguinte forma:

	Memória Lógica Processo 1			Memória Física Verdadeira		Memória Lógica Processo 2
11	Pilha SO		11		11	Pilha SO
10			10	Dados SO	10	
9	Dados SO		9	Pilha SO	9	Dados SO
8	Código SO		8	Pilha P1	8	Código SO
7	Pilha P1		7	Código SO	7	Pilha P2
6			6	Dados P2	6	
5			5	Código P1	5	
4			4	Dados P1	4	
3			3	Pilha P2	3	
2	Dados P1		2	Código P2	2	Dados P2
1	Código P1	4095	1	Código P1	1	
0		0	0	Dados P2	0	Código P2

Na Memória Lógica do Processo existe um limite para colocar as áreas do processo, pois na parte de cima vai ficar o SO. No exemplo ao lado, o limite para ter conteúdo do processo na memória lógica é a página lógica 7. O SO fica nessa parte em qualquer processo. Sendo o mesmo SO em ambos os processos. Isso é possível porque as paginas logicas que estão nos processos apontam para as mesmas paginas físicas do SO.

Este é um artificio que se faz para dar a impressão que o SO esta presente em toda memoria logica. Na memoria verdadeira que é a memoria física o SO está somente em algumas paginas. Esse é o mesmo mecanismo que é utilizado na DLL (Biblioteca compartilhada).

Com TLB "Simples"

TLB do Processo 1			
Num_pag_logica	Num_pag_Fisica	Executável	Alterável
0	1	1	0
2	4	0	1
8	7	1	0
9	10	0	1

O SO executa o processo 1 então guarda na TLB as conversões do processo 1. Conforme mostra a Figura da TLB "Simples" ao lado.

A TLB é preenchida de acordo com a execução do processo 1. Este processo faz uma chamada ao Sistema Operacional e não será preciso esvaziar a TLB, pois o SO está na memória lógica do processo. A memória lógica não muda quando ocorre uma chamada ao SO ou uma interrupção, pois o processo é o mesmo e seus dados estão na TLB.

Esta solução é a mais utilizada. A maior parte dos Sistemas Operacionais trabalha assim. Windows e o Linux trabalham com esta opção, pois não gera perda de desempenho. A Intel trabalha assim. A TLB só é esvaziada quando troca o processo ativo e a memória lógica muda. A TLB existe sempre que existir paginação. Ela é controlada pelo HW e não pelo SO.

Vantagens do SO estar na memória lógica de todos os processos

- Essa solução do SO está na memoria logica do Processo funciona bem com a TLB "Simples".
-
-

Desvantagens do SO estar na memória lógica de todos os processos

-
-
-

Vantagens da Paginação

- Proteção da memória, pois um processo não acessa as paginas de outro processo;
- Não existe mais o problema de carregamento do processo na memória, pois todo processo começa no endereço zero;
- Conseguir colocar dois processos na memória verdadeira sem problemas de um processo confundir sua memória com a memória de outro processo, pois o processo acha que está sozinho na memória física só que isto só é verdade na memória lógica;

- Não existe Fragmentação Externa;
- Poder ter várias memorias logicas;
- Enquanto existir espaço disponível na memória poderá se alocado um espaço de endereçamento lógico para outros processos. Assim é possível aumentar o tamanho da pilha ou criar novas áreas como, por exemplo, a Biblioteca.

Desvantagens da Paginação

- Número inteiro de páginas por áreas. Por exemplo, o tamanho da página (4 Kb). Podemos ter um código que precise apenas de 6 KB. Logo, 2 Kb serão desperdiçados sendo chamados de Fragmentação Interna. Este é um problema da paginação que não tem solução, pois o desperdício é muito pequeno comparado com o ganho que se tem com paginação;
- Ter que acessar a memoria física duas vezes: uma para fazer a conversão do endereço logico para o fisico e outra para acessar o dado e jogar para o registrador. Isso deixa a CPU duas vezes mais lenta;
- A tabela de página ficar na memória, pois pode ser grande e não caber no chip da CPU.
- A Tabela de Página é muito grande
- Perda de desempenho na conversão do endereço logico para o endereço físico.

Observações

- Os processos tem que ser contínuos no mundo virtual. É possível aumentar a pagina na memória virtual, basta existir paginas reais na memória real. O programa vê como única o espaço de endereçamento virtual.
- Cada memória lógica tem o seu endereço zero. Na hora que o compilador está compilando para linguagem de máquina ele pode assumir que o endereço inicial do processo sempre será o endereço zero. Então todo o código fonte que é compilado é compilando com esta suposição.
- A regra de ter que alocar uma área continua na memória isto é verdade na memória lógica, mas não é verdade na memória física. As áreas do processo precisam estar continua no mundo ilusório. No mundo ilusório nunca vai ter fragmentação, pois existem milhares de páginas lógicas entre as áreas de pilha e a área de dados. Se estas áreas precisarem crescer vão poder.
- O somatório das paginas logicas que tem conteúdo tem que ser menor ou igual às paginas da memória física da máquina.

19) PAGINAÇÃO SOB DEMANDA

É uma forma melhor de evitar a falta de memória física. Também chamada de Memória Virtual (mas memória virtual já existia antes deste mecanismo) ou Paginação com o uso do Disco.

A paginação em disco ocorre quando a memória física fica cheia, ou seja, não tem pagina física livre na memoria verdadeira. Então não tem como colocar um novo processo ou criar uma nova pagina na memoria logica do processo por uma razão qualquer, por exemplo, crescer a área de pilha do processo 1 ou criar uma Biblioteca compartilhada.

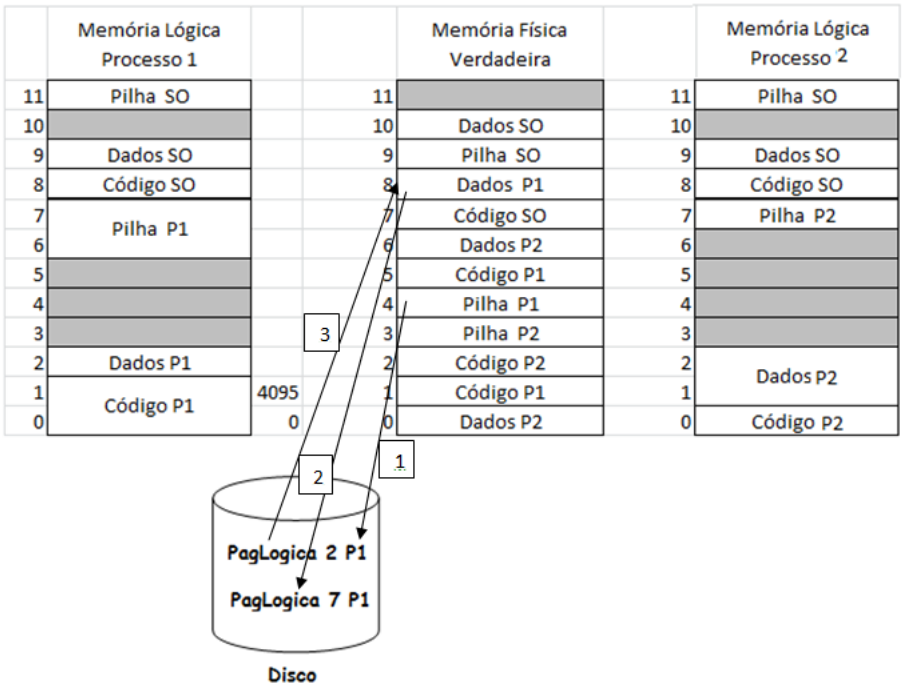
Na pilha existe o topo (é um registrador) tudo que está acima do topo está ocupado e o que está abaixo do topo está livre. Conforme as variáveis são empilhadas o topo abaixa. Pode acontecer de o topo bater no limite inferior da pilha. Nesse caso o HW gera interrupção e pagina abaixo da área de pilha é válida. Essa interrupção é especial e o SO vai tentar aumentar o topo da pilha, mas só será possível aumentar o tamanho da pilha se tiver memoria física livre.

Nesse caso especifico não existe memoria física livre. Logo, não é possível de forma simples aumentar o tamanho da pilha do processo 1. Vai dar erro no processo o SO não conseguiu aumentar o tamanho da área de pilha. Para resolver esse problema a paginação precisa fazer o uso do disco.

Então quando a memoria física está cheia e precisa de mais paginas físicas livres. Entra em ação um mecanismo do SO para tirar paginas da memoria física e salvar em disco para liberar espaço na memoria física.

O nome da interrupção gerada quando não tem memoria livre é Falta de Pagina (Page Fault).

EXPLICAÇÃO DO FUNCIONAMENTO DA PAGINAÇÃO SOB DEMANDA



Conforme mostra a Figura acima no **passo 1** o SO escolhe a pagina física 4 para ser salva no disco. O conteúdo desta pagina é a pagina logica 2 do processo 1. Assim área que a pagina física 4 ocupava na memoria física passa a ficar livre.

Tabela de Páginas do Processo 1

Num_pag_logica	Num_pag_Fisica	Válido	Executável	Alterável
11	9	1	0	1
10	-	0		
9	10	1	0	1
8	7	1	1	0
7	8	1	0	1
6	-	0		
5	-	0		
4	-	0		
3	-	0		
2	4	1 0	0	1
1	5	1	1	0
0	1	1	1	0

A página logica 2 do processo 1 estava na pagina física 4 e o bit válido era 1. Quando acontece estouro da pilha. O SO tenta aumentar o topo da pilha e não consegue. Então ele escolhe uma página vitima para salvar em disco. Nesse exemplo, a pagina física 4. No **passo 2** o SO salva esta pagina física no disco ela não está mais valida. Então o campo bit Valido é alterado para 0 conforme mostra a Figura da Tabela de Páginas do Processo 1. Assim o SO consegue arrumar um espaço na memoria física para aumentar a pilha do processo 1.

Esta mesma solução é aplicada para resolver o problema de criação de um processo novo. Nesse caso pode ser necessário salvar mais de uma pagina física no disco para liberar espaço na memoria física e colocar as paginas logicas do novo processo na memória física.

O problema da falta de pagina ocorreu e foi resolvido pelo SO não apresentando nenhum erro para o usuário. A pagina logica 2 do processo não está mais na memoria física. Neste momento ela é uma pagina invalida. Se o processo 1 quiser usar esta pagina logica, por exemplo, executa a instrução → MOV AH, [8200].

Quando o HW tenta fazer a conversão do endereço logico para o endereço físico ele consultará primeiro a TLB. Não encontrando a página nela o HW consulta a Tabela de Páginas, mas dará erro, pois a página lógica 2 não está na memória física. Então nesse momento o HW gera interrupção, pois a página lógica está com o bit válido igual 0 na tabela de páginas, ou seja, não está na memoria física.

A mesma interrupção é gerada se o processo tentar usar a pagina logica 4, pois não é uma pagina logica válida. Conforme mostra a Figura da Tabela de Páginas do Processo acima um caso é diferente do outro. Tentar usar a pagina logica 4 é um erro do processo, pois não tem nenhum conteúdo e tem que abortar o processo. Por ouro lado, tentar utilizar a pagina logica 2 não é um erro, pois tem conteúdo nela. Ela só está invalida porque foi salva em disco.

Logo, no **passo 3** quando o SO vai tratar a interrupção da pagina logica 2 (que está em disco) esta página é trazida do disco para memória.

Mapa de Memória do Processo 1		
Tipo	Pagina Inicial	Qtd Páginas
Codigo	0	2
Dados	2	1
Pilha	6	2

A Figura ao lado mostra o Mapa de Memória do Processo o SO consulta esta tabela para saber as páginas lógicas que estão em disco, pois nessa tabela fica registrado quantas paginas cada área do processo tem se tiver faltando na Tabela de Página é porque foi salva no disco para liberar espaço na memória física.

Ao consultar a Tabela com o Mapa de Memória do Processo o SO sabe que a pagina logica 2 pertence a área de dados do processo 1. E a pagina logica 4 não pertence ao processo, então o tratamento da interrupção gerado por essa pagina é para abortar o processo. Mas, o tratamento de interrupção da pagina logica 2 é para colocar de novo na memoria esta pagina.

Tabela de Páginas do Processo 1				
Num_pag_logica	Num_pag_Fisica	Válido	Executável	Alterável
11	9	1	0	1
10	-	0		
9	10	1	0	1
8	7	1	1	0
7	8	~~1~~0	0	1
6	4	1	0	1
5	-	0		
4	-	0		
3	-	0		
2	~~4~~8	~~1~~0 1	0	1
1	5	1	1	0
0	1	1	1	0

Como a memoria física continua cheia para trazer a pagina logica 2 para memoria e preciso aplicar o mecanismo outra vez para escolher outra pagina vitima. Conforme mostra o passo 3 o SO escolheu a pagina física 8. Agora esta pagina está com o conteúdo da pagina logica 2. O SO ajusta na tabela de paginas o novo numero da pagina física da pagina logica 2 e altera o bit invalido para valido dessa pagina. Conforme mostra a Tabela de Páginas do Processo ao lado.

O tratamento da interrupção termina, a instrução que gerou a interrupção é reiniciada. Mas, dessa vez a conversão do endereço logico no endereço físico é realizada com sucesso. Tudo isso ocorreu sem apresentar erro para o usuário, pois funciona com transparência.

Enquanto a memoria física tiver cheia ocorrerá trocas de páginas. Este mecanismo também é chamado de SWAP de páginas. Ele é parecido com o SWAP de processos a diferença do SWAP de páginas para o SWAP de processos é o que vai para o disco não é o processo inteiro, mas sim uma pagina logica do processo.

Nesse exemplo, do estouro de pilha, no momento que o SO trata a interrupção gerada na execução do processo 1. O processo 1 para de executar. A interrupção desse exemplo fez duas coisas salvou a pagina vitima (pagina logica 7) e depois teve que mandar lê página logica 2. Foi feito dois acessos ao disco durante o tratamento dessa interrupção.

O acesso ao disco é muito lento. O tempo que demora em ler o conteúdo do disco dá para executar milhões de instruções. Como é importante não desperdiçar a CPU enquanto o disco lê o conteúdo. Em uma situação como essa o SO escalonará outro processo para executar. Sempre que o processo precisar lê um arquivo no disco, o SO bloqueia o processo e coloca outro processo em execução.

Esse mecanismo possui os seguintes momentos:

1) A memória física está cheia;
2) Um processo precisa de uma pagina física;
3) Para abrir espaço na memoria física o SO escolhe uma pagina vitima para salvar no disco;
4) O SO salva a página vitima no disco e a memoria física fica com uma pagina livre;
5) O SO salva a pagina do processo no lugar da pagina vitima;
6) Em um dado momento o processo que contem a pagina vitima precisa dessa pagina;
7) O SO verifica se a memoria continua cheia. Se tiver ele escolhe outra pagina vitima e salva no disco;
8) O SO traz a pagina vitima do disco para memoria;
9) O processo dono da pagina vitima pode continuar sua execução.

Portanto, é importante que o SO escolha bem as páginas vítimas de forma que elas não sejam usadas no futuro pelo processo, essa é a escolha ideal, pois a troca constante de disco para a memória pode deixar a máquina lenta. Hoje em dia, com o aumento do tamanho da memoria física esse mecanismo funciona muito bem. Ele é transparente tanto para o usuário quanto para o próprio processo.

PÁGINA VITIMA

A página vitima é a pagina física escolhida para ser colocada em disco. Na Tabela de Paginas a entrada correspondente a pagina escolhida como vitima, passa a ter o bit Válido = 0, e o campo NumPagFisica correspondente fica vazi para indicar que a pagina foi para disco, ou seja, não está na memória. O SO guarda as páginas que foram salvas em disco em uma Tabela Auxiliar.

A escolha da Pagina Vitima é feita através dos seguintes algoritmos: FIFO, ÓTIMO, LRU, NRU, 2ª CHANCE E RELOGIO.

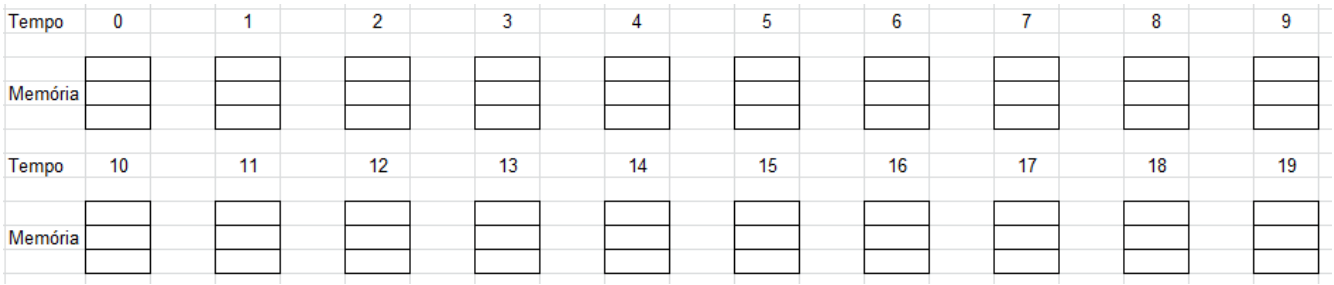
ALGORITMOS DE ESCOLHA DE PAGINA VÍTIMA

A seguir temos a sequencia temporal de paginas logicas que um processo acessou:

O SO escolhe as paginas que vão sair da memória física, ou seja, as paginas vitimas através dos algoritmos de escolha da pagina vitima. Veremos como funciona estes algoritmos a seguir. Utilizaremos no exemplo uma memoria física com 3 paginas e um processo com 8 paginas logicas.

FIFO (FIRST IN, FIRST OUT)

Nesse algoritmo a página que entrou primeiro (mais antiga) é a primeira a sair. Então, de acordo com a sequencia temporal das paginas logicas acessadas pelo processo apresentada acima a representação desses acessos fica da seguinte forma:



Temos ___ interrupções que é igual ___ Page Fault (Falta de Página).

Conforme mostra a Figura acima conseguiremos colocar na memória física sem termos problema de Falta de Pagina até a terceira pagina logica do processo. Ao tentar colocar a quarta pagina logica que é a pagina logica 2 na memoria física não conseguiremos, pois ela está cheia.

A pagina logica 3 que não está na memoria física é uma pagina logica inválida na Tabela de Paginas do Processo. O HW gera uma interrupção quando o processo tenta acessar está página. Então o SO ao tratar essa interrupção identifica que a página está salva no disco e precisa trazer esta página para a memoria física. Como a memoria física está cheia o SO escolhe uma pagina vitima entre as paginas que estão na memória de acordo com o Algoritmo FIFO.

Vantagem do FIFO

- É o algoritmo com implementação mais simples, pois a página escolhida como vítima é sempre aquela que está há mais tempo na memória.
-
-

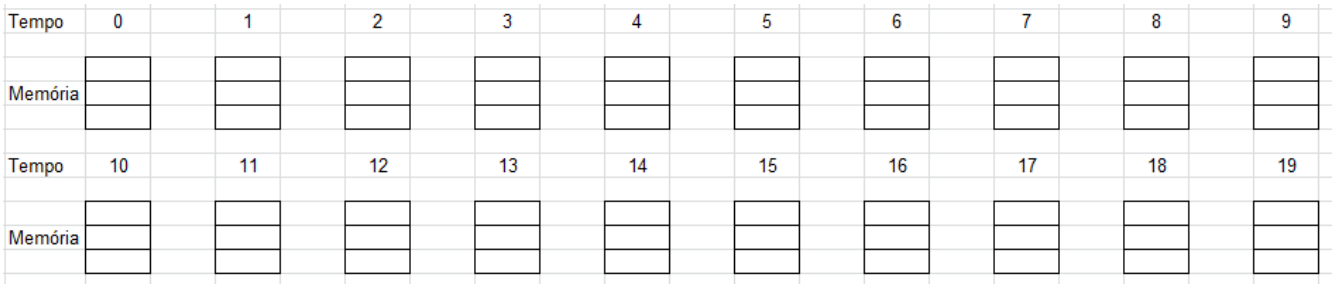
Desvantagem do FIFO

- O desempenho do FCFS não é bom, pois o fato de uma página estar velha na memória principal não está relacionado com a sua necessidade (ou não) para a execução do processo.
-
-

ÓTIMO (ALGORITMO TEÓRICO)

Esse algoritmo olha o futuro e vê qual a página que só será usada muito mais tarde e que deverá ser substituída a seguir. Infelizmente esse algoritmo não pode ser usado pelo SO porque simplesmente eles não podem adivinhar o futuro.

No tempo T0 as três posições encontram-se vazias. Como do T0 ao T2 há espaço para mais uma página, elas vão sendo alocadas no primeiro espaço livre encontrado. No tempo T3 a página 2 precisa ser alocada, mas não há espaço livre na memória. Utilizando o algoritmo ótimo, a página que dará vez à página 2 será a página 7, pois só será usada novamente no tempo 17 ao contrário das demais que serão usadas novamente muito antes.



Temos ___ interrupções que é igual ___ Page Fault (Falta de Página).

Contudo, é impossível implementar tal algoritmo, pois ele exige o conhecimento prévio do comportamento dos processos. Saber quais páginas um processo vai acessar no futuro implica conhecer exatamente o seu fluxo de controle futuro. Como, na maioria dos programas, o fluxo de execução depende dos dados processados, não é possível implementar o algoritmo ÓTIMO. Entretanto, podemos tentar imaginar algoritmos factíveis que aproximem o seu comportamento.

Vantagens do Ótimo

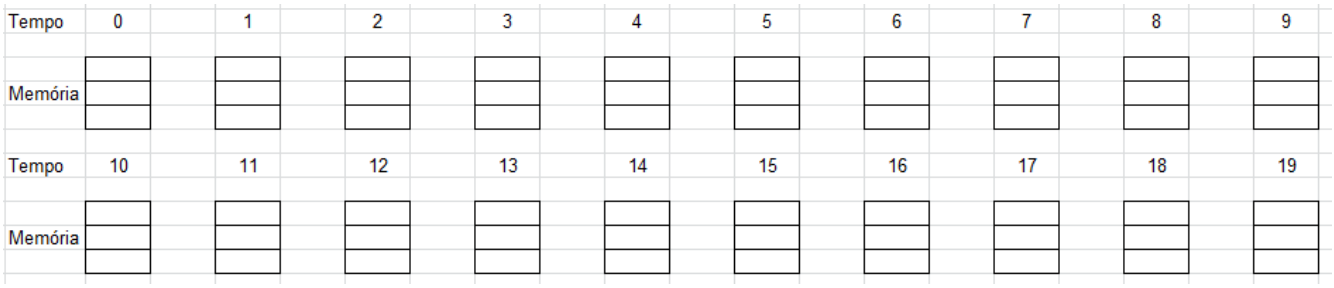
- Gera muito menos interrupção, pois se elimina o problema de uma página que foi substituída e que logo depois precisa ser usada novamente.
-
-

Desvantagens do Ótimo

- Precisa conhecer o Futuro, mas não tem como o SO saber o futuro. Logo é um algoritmo somente teórico servindo apenas como base de comparação.
-
-

LRU (LEAST RECENTLY USED – MENOS USADA RECENTEMENTE)

Como o SO não pode olhar o futuro, ele olha o passado para tentar prever o futuro, usando o algoritmo LRU que substitui as páginas que foram usadas há mais tempo, ou seja, escolhe a pagina que teve o uso mais distante no passado. A página vitima é a página menos usada recentemente.



Temos ___ interrupções que é igual ___ Page Fault (Falta de Página).

No tempo T0 as três posições encontram-se vazias. Como do T0 ao T2 há espaço para mais uma página, elas vão sendo alocadas no primeiro espaço livre encontrado. No tempo T3 a página 2 precisa ser alocada, mas não há espaço livre na memória. Utilizando o algoritmo LRU, a página que dará vez à página 2 será a página 7, pois foi usada há mais tempo.

Em 9 a pagina logica 0 tem que entrar na memoria física, mas o Algoritmo LRU acabou de escolher ele como página vitima é salvar no disco. Nesse caso o algoritmo não fez uma boa escolha. O problema é que para o LRU escolher a pagina vitima tem saber qual foi a sequencia exata de paginas acessadas pelo processo, mas isso não é facilmente conhecido porque o SO, por exemplo, não sabe quais paginas foram para memoria física ele só fica sabendo das coisas quando está em execução.

Portanto, a sequencia exata embora ela tenha ocorrido ninguém fica sabendo no futuro. Embora o algoritmo LRU seja possível ser implementado na prática, hoje em dia, ele não é porque não tem a informação disponível para o SO.

Vantagens do LRU

- O algoritmo LRU em geral é melhor que o algoritmo FIFO.
-
-

Desvantagens do LRU

- É pior que o algoritmo Ótimo.
-
-

NRU (NOT RECENTLY USED – NÃO USADO RECENTIMENTE)

O NRU é uma alternativa de verificar o tempo de uso das páginas. Derivado do LRU, ele utiliza a ideia do uso recente, mas mantendo a ordem exata de como a página foi usada. Isto é feito através de mais um campo na Tabela de Páginas que é o bit "Acessado". Toda vez que uma página lógica é acessada o HW muda esse bit para 1.

Tabela de Páginas do Processo 1

Num_pag_logica	Num_pag_Fisica	Válido	Executável	Alterável	Núcleo	Alterado (Dirty = Sujo)	Acessado (ou usado)
4							
3							1
2							0
1							0
0							

O Bit Acessado (ou Usado) só diz se teve ou não acesso a pagina logica, mas não diz se foi recente. Ao consultar esse bit o SO consegue descobrir quais paginas foram acessadas, mas o SO não sabe que o acesso a pagina foi recente. Para isto é implementado um mecanismo para auxiliar o SO.

Esse mecanismo de tempos em tempos altera o bit Acessado para 0. Assim, o SO consegue saber qual a pagina logica que teve o uso mais recente, ou seja, é pagina logica com o bit Acessado igual 1 (ainda). Isto quer dizer que esta pagina foi acessada depois da ultima vez que o mecanismo do SO zerou o campo bit Acessado.

O HW que não tem o Bit Acessado o SO simula o Bit Acessado para auxilia-lo a descobrir quando a pagina logica foi acessada pela ultima vez. Conforme mostram as Figuras abaixo.

Tabela de Páginas do Processo sem o Bit Acessado

Num_pag_logica	Num_pag_Fisica	Válido	Executável	Alterável	Núcleo
7		0			
6					
5					
4		~~1~~ 0 1			
3		0			
2		~~1~~ 0			
1		~~1~~ 0			
0		~~1~~ 0			

Tabela Auxiliar usada somente pelo SO

Num_pag_logica	Realmente Válido	Acessado
7	0	0
6		
5		
4	1	~~0~~ 1
3	0	0
2	1	0
1	1	0
0	1	0

Digamos que estejam validas as paginas 0, 1, 2 e 4. O SO simula o bit Acessado colocando uma informação falsa na tabela de paginas do processo. Ele coloca Valido igual 0 nas paginas logicas que estavam validas. O SO coloca a informação correta na tabela auxiliar que usada somente pelo SO no campo Realmente Valido que vão conter o valor 1. No campo Acessado ele coloca o valor 0.

Quando o processo precisar utilizar, por exemplo, a pagina logica 4 o HW vai gerar uma interrupção. O SO vai tratar essa interrupção e verifica o campo "Realmente Válido" na Tabela Auxiliar o valor verdadeiro do bit Válido da Tabela de Pagina do Processo. Logo, o SO não aborta o processo e no tratamento dessa interrupção ele liga o bit Acessado e coloca o valor correto no bit Válido na pagina logica 4 da Tabela de Pagina do Processo. Conforme mostram as Figuras acima.

De tempos em tempos uma rotina é executada e zera todos os bits de todas as páginas. Digamos que falte memória e o SO tenha que escolher uma página vítima. Ele escolherá a página que está com bit Acessado = 0, pois essa página foi acessada recentemente. Sabe-se, com isso, quem teve acesso recente, e quem não teve acesso recente. Então, escolhe uma das páginas que não teve acesso recente.

Portanto, esse algoritmo se baseia na suposição de que as páginas acessadas recentemente por um processo continuarão sendo acessadas por ele no futuro próximo. A utilização de laços, sub-rotinas e módulos na construção do programa faz

com que essa suposição seja verdadeira na maior parte das vezes. Embora possível de ser implementado, o NRU exige um suporte de hardware raramente encontrado: a MMU precisa manter na tabela de páginas o instante exato no qual cada página foi acessada pela última vez. Isso implica em manter um registrador adicional de vários bytes para cada uma das entradas da tabela.

O NRU poderia ser feito, pois não é teórico. Ele usa o passado. Só que o problema é ter que saber a sequencia de paginas que foram executadas corretamente, mas o SO não sabe isso. Logo esse algoritmo não consegue ser executado. Se ele soubesse seria utilizado o NRU.

Vantagens do NRU

-
-
-

Desvantagens do NRU

-
-
-

2ª CHANCE (OU RELÓGIO)

Esse algoritmo é parecido com o algoritmo FIFO só que dá uma segunda chance para a pagina caso a mesma já tenha sido acessada. A segunda chance é feita colocando a pagina no final da fila de novo e alterando o Bit Acessado para 0.

Aplicando o algoritmo da 2ª Chance na sequencia de paginas logicas utilizadas pelo processo temos o seguinte: a pagina logica 7 é colocada na memoria física e o bit Acessado dela é alterado pelo HW para 1. Colocamos a pagina logica 0 na memoria e o bit Acessado dela igual a 1. Depois colocamos a pagina logica 1 e o bit Acessado dela igual a 1. A memoria encheu precisamos tirar uma pagina logica da memoria para salvar em disco e colocar a próxima pagina logica que é a 2.

Tempo	0	1	2	3	4	5	6	7	8	9
Memória										
Tempo	10	11	12	13	14	15	16	17	18	19
Memória										

Temos ___ interrupções que é igual ___ Page Fault (Falta de Página).

A pagina vitima é a primeira pagina logica que foi para memória física. Nessa hora é avaliado se a pagina 7 terá uma segunda chance. Nesse caso terá, pois o bit Acessado dela está igual a 1. Ela é colocada no final da fila novamente e seu o bit Acessado é alterado para 0. No lugar dela é colocada a pagina logica 2.

Esse procedimento é repetido até que todas as paginas logicas da sequencia apresentada tenham ido para memoria física na ordem apresentada. Lembrando que a pagina vitima só terá segunda chance se o bit Acessado dela estiver igual a 1 quando ela foi escolhida como pagina vitima.

Se a pagina logica estiver com o bit Acessado igual a 0 ela será a pagina vitima. Se estiver com esse bit igual 1 não será a pagina vitima e terá a segunda chance.

Este algoritmo é utilizado no Windows. Ele segue a lógica do algoritmo NRU. O Windows mantem uma fila com a ordem que as paginas vieram da memoria. No caso do Linux é igual ao numero da pagina a pagina seguinte é igual ao numero da pagina que se tem o nome desse algoritmo é Relógio.

Vantagens da 2ª Chance

-

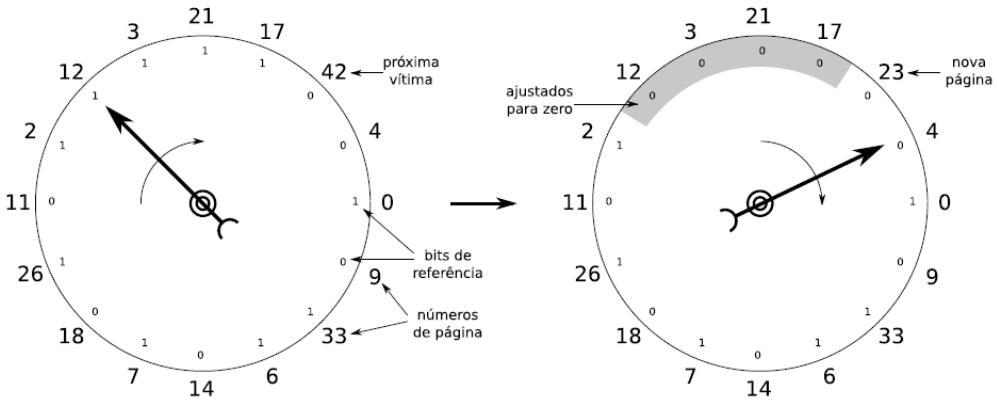
-
-

Desvantagens da 2ª Chance

- Tem muita página e se zerar todos os bits de tempo em tempo terá perda de Desempenho.
-
-

RELÓGIO

Um ponteiro percorre a fila sequencialmente, analisando os bits de referência das páginas e ajustando-os para zero à medida em que avança. Quando uma página vítima é encontrada, ela é movida para o disco e a página desejada é carregada na memória no lugar da vítima, com seu bit de referência ajustado para zero. Essa implementação é conhecida como algoritmo do relógio e pode ser vista na figura abaixo.



Algoritmo do relógio

Quando uma página vítima é necessária, o algoritmo verifica o bit de referência da página indicada pelo apontador. Caso esse bit esteja desligado, essa página é efetivamente escolhida como vítima, e o apontador avança uma posição na lista circular.

Caso o bit de referência da página apontada esteja ligado, o bit de referência é desligado, e ela recebe uma segunda chance. O apontador avança uma posição na lista circular, e o procedimento é repetido para a próxima página. No caso extremo, todas as páginas possuem o bit de referência ligado.

Nesse caso, o apontador fará uma volta completa na lista, desligando todos os bits de referência. Ao chegar novamente na primeira página visitada, essa terá agora o bit de referência desligado e será escolhida como página vítima. Devido a esse comportamento circular, o algoritmo também é conhecido como algoritmo do relógio (clock algorithm), em analogia aos ponteiros de um relógio.

Para começar estão todos zerados, um ponteiro vai passando e só anda quando tem falta de página. O ponteiro vai zerando os que não estiverem zerados. Divide páginas em páginas que tiveram acessos recentes e páginas que não tiveram acessos recentes. Se a página não foi executada desde a última vez que o ponteiro passou por ela é porque não teve uso recente. Se a página não foi acessada não teve uso recente. Escolhe um tempo de passagem e considera recente se não passou nesse tempo. Assim, se for um processo antigo pode ser vítima.

Portanto, de tempos em tempos o algoritmo zera tudo, então o acesso recente é o tempo que ele zerou desde a última vez que ele zerou todo mundo. É o momento que o ponteiro passou pela página pela última vez, se não teve uso nesse intervalo de tempo é por que não teve uso recente e a pagina pode ser escolhida como vitima.

O Relógio é similar ao da 2ª Chance, mas não tem fila. Para saber se a pagina terá ou não uma 2ª chance é utilizado um ponteiro que percorre a sequencia de paginas numéricas 0,1, 2, 3, 4 etc. A pagina que estou apontando agora foi acessada? Não. Então não será a pagina vitima. Se a pagina seguinte não foi acessada ela será a vitima. A ideia é manter na memoria as paginas que foram acessadas, ou seja, estão com o bit acessado ligado.

Vantagens do Relógio

-
-
-

Desvantagens do Relógio

-
-
-

ANOMALIA DE BELADY

Diz que a quantidade de falta de páginas pode aumentar quando o tamanho da memória também aumenta.

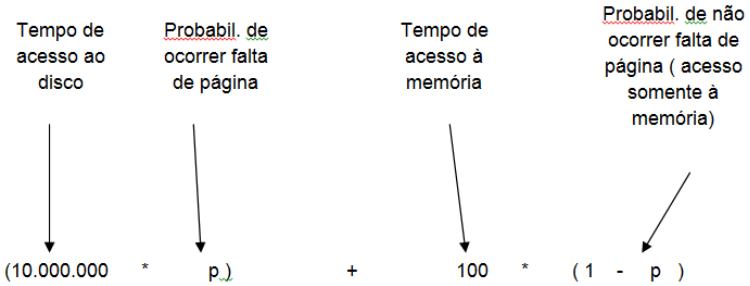
O algoritmo FIFO apresenta essa anomalia.

O LRU corrige o problema dessa anomalia de Belady através do conceito de pilha (Retira sempre a página que está no topo da pilha, que é a página usada há mais tempo). Assim, o aumento no tamanho do conjunto residente implica apenas no aumento do tamanho da pilha.

O 2ª chance também utiliza o conceito de pilha (Fila de páginas, sendo que a página que sairá da memória deve ser uma que tenha pouco uso e não tenha sido alterada). Logo, não possui essa anomalia.

ANÁLISE DO DESEMPENHO DE ACESSO A MEMÓRIA NA PAGINAÇÃO SOB DEMANDA

Tempo de acesso à memória = 100 ns
Tempo de acesso ao disco = 10 ms = 10.000.000 ns



Devido ao grande tempo de acesso ao disco, a falta de página não pode ocorrer com frequência. Se a probabilidade de ocorrer falta de página for de 0,0001%, temos:

(10.000.000 * 0,000001) + 100 (1 – 0,000001) = 100 + 99,9999 ≈ 200 ns

O problema nesse caso é o acesso a memoria porque a pagina logica acessada pode estar em disco ou não. Então temos três casos de acesso a memoria logica:

- ✓ Pagina Lógica válida (Bit válido = 1).
- ✓ Página Lógica inválida (Bit válido = 0), mas está em disco.
- ✓ Pagina Logica invalida que não pertence ao processo. Caso de erro o SO aborta o processo.

No caso do desempenho o que interessa são os dois primeiros casos que não abortam.

No caso 1 temos: Tempo Físico é igual a aproximadamente o tempo de acesso à memória que é igual aproximadamente a 100 ns = 10^{-7} s (Tempo de Acesso a Memória)

No caso 2 é mais complicado. Temos: Tempo é igual aproximadamente o Tempo de leitura da página que é igual aproximadamente a 10 ms = 10^{-2} s (Tempo de Acesso ao Disco).

Observação: 1 ms = 10^{-3} s e 1 ns = 10^{-9} s

O tempo de acesso ao disco é 100 mil vezes mais lento do que o tempo de acesso a memória. Isso dá uma perda de desempenho brutal quando tem que acessar o disco. Colocando isso em uma formula só temos o seguinte:

Tempo Médio de acesso a Memoria Logica = ($Px10^{-7}$) + (($1-P)x 10^{-2}$)

Onde: P = Probabilidade da Pagina Logica estar na memoria física.

Estipulando a perda de desempenho de 20%

Tempo médio é 20% maior que o caso 1 = 1,2 x 10^{-7}

1,2 x 10^{-7} = ($Px10^{-7}$) + (($1-P)x 10^{-2}) → P = 99,9998 %

O que podemos mostrar é que a perda de desempenho é grande com a paginação sob demanda. Para ser pequena a perda de desempenho na maioria dos casos ((99,9998 %) a pagina logica tem que estar na memória física. E somente em uma minoria dos casos (00,0002%) a pagina logica estar no disco.

É muito importante que poucas vezes o SO traga a pagina logica para memória, pois se isto acontecer com muita frequência à perda de desempenho será muito grande. O uso do disco quando a memoria física está cheia introduz uma perca de desempenho muito grande. Nesse caso só foi considerando o tempo de um acesso ao disco que é o acesso de leitura.

No exemplo dado acima temos que considerar o tempo do SO salvar o conteúdo de uma pagina física no disco e depois trazer a pagina logica que está no disco para a memória física para então o processo utiliza-la. Nesse caso que é o caso 2 o tempo será multiplicado por 2. Sendo duas vezes pior.

Portanto, o SO precisa escolher bem qual página deverá ser colocada em disco (página vítima) de forma que esta não seja usada brevemente pelo processo e, consequentemente, não gaste mais tempo fazendo as trocas de páginas entre memória e disco. O SO olha o que ocorreu para prever o que poderá acontecer, usando o Princípio da Localidade.

PRINCÍPIO DA LOCALIDADE

Os processos tendem a usar as mesmas páginas em um determinado instante de tempo. Explicado pela estrutura em que o código é montado (rotinas com suas variáveis locais). Assim, o SO tenta manter em memória as páginas usadas há pouco tempo no passado, pois a tendência de se usar estas páginas é maior do que aquelas usadas num passado distante.

OTIMIZAÇÕES - COMO EVITAR O USO DO DISCO

Tem varias ocasiões que não precisa utilizar o disco e o SO vai utiliza-las, pois ele deseja fazer somente o acesso de leitura. A seguir algumas dessas ocasiões (mecanismos):

1) Paginas de código quando são paginas vitimas não são salvas no disco, pois já estão no disco dentro do arquivo executável. Como o código não pode ser modificado enquanto está na memória, ele é exatamente uma cópia do que está em disco. Mas, para isto acontecer não será permitido apagar o arquivo executável que estiver em execução.

2) Quando o SO escolhe como vitima uma pagina que já foi vitima anteriormente e não foi alterada desde então não é necessário salvá-la. Páginas alteráveis (tipicamente as que contêm variáveis) que foram vitimas anteriormente e não foram alteradas desde então (se forem vitimas novamente) não são salvas novamente, pois o conteúdo é o mesmo. Dificuldade como saber se uma pagina foi alterada ou não?

3) O SO carrega as páginas lógicas de código Sob Demanda.

4) "POOL" de páginas livres: O SO nunca deixa que a memória fique completamente cheia. Então ele pré-salva a página vítima liberando espaço na memória.

Solução da Dificuldade encontrada na Solução 2 apresentada acima

O SO não sabe tudo que está acontecendo. Ele só tem capacidade de fazer coisas quando ele entra em execução. O SO entra em execução quando o processo faz uma chamada ao SO ou quando ocorre uma interrupção. Se isto não acontecer o SO não sabe de nada. Sendo assim, não é trivial ele saber, mas ele pode saber com ajuda do HW.

O HW ajuda o SO descobrir qual a página alterada através do bit Alterado (Dirty = sujo) da Tabela de Paginas do Processo. Todos os bits citados até esse momento é o SO que altera e o HW consulta. Este é o SO que consulta e o HW que altera.

Por exemplo, a pagina logica 2 foi a pagina vitima escolhida para ser salva no disco e depois voltou para memoria física. Depois que ela volta para memoria o SO coloca neste bit (Alterado) o valor 0 conforme mostra a Figura ao lado para dizer que a pagina não foi

Tabela de Páginas do Processo 1

Num_pag_logica	Num_pag_Fisica	Válido	Executável	Alterável	Núcleo	Alterado (Dirty = Sujo)
11	9	1	0	1	1	
10	-	0				
9	10	1	0	1	1	
8	7	1	1	0	1	
7	8	1	0	1	0	
6	4	1	0	1	0	
5	-	0				
4	-	0				
3	-	0				
2	4	~~1~~ 0	0	1	0	~~0~~ 1 0
1	5	1	1	0	0	
0	1	1	1	0	0	

Então essa ajuda do HW permite que o SO saiba se a pagina logica foi alterada ou não. Se for 0 não precisa ser salva novamente no disco, pois nada foi alterado. Se for 1 precisa salvar novamente no disco, pois o que está no disco está desatualizado. A Intel trabalha com este bit outros fabricantes não tem esse bit na tabela de paginas.

A Intel possui a Tabela de Paginas do Processo sem o bit Alterado. Nesse caso o HW que não tem esse bit o SO faz a simulação desse bit com a ajuda de uma Tabela Auxiliar. O SO engana o HW colocando uma informação falsa na Tabela de Paginas, ou seja, se a página é Alterável ele seta o bit Alterável para 0 e colocar a informação verdadeira no campo Realmente Alterado da Tabela Auxiliar.

Tabela de Páginas do Processo 1

Num_pag_logica	Num_pag_Fisica	Válido	Executável	Alterável	Núcleo
11	9	1	0	1	1
10	-	0			
9	10	1	0	1	1
8	7	1	1	0	1
7	8	1	0	1	0
6	4	1	0	1	0
5	-	0			
4	-	0			
3	-	0			
2	8	1	0	~~1~~ 0 1	0
1	5	1	1	0	0
0	1	1	1	0	0

Tabela Auxiliar usada somente pelo SO

Num_pag_logica	Realmente Alterado	Alterado
11		
10		
9		
8		
7		
6		
5		
4		
3		
2	1	~~0~~ 1
1		
0		

Conforme mostra as Tabelas ao lado, o SO diz que a pagina logica 2 não é alterável (coloca 0) na Tabela de Pagina.

Na Tabela Auxiliar no campo Realmente Alterável coloca 1 (valor verdadeiro) e no campo Alterado coloca 0 (valor Falso alterado pelo SO).

Desse jeito dá para simular o bit Alterável porque na hora que o processo tentar alterar a pagina logica 2. O que o HW vai dizer que não pode, pois na Tabela de Página do Processo essa pagina logica é não alterável. Logo, o HW gera uma interrupção. Quando o SO vai tratar essa interrupção, ele verifica na Tabela Auxiliar se essa informação está correta e descobre que é falsa. Ou seja, deveria ser Alterável = 1. Então o SO utiliza a interrupção para mudar o valor do bit Alterado para 1, ou seja, coloca o bit Alterável igual a .

O SO faz três coisas em uma pagina que voltou ao disco no caso que o HW não tem o bit Alterado que são:

1) Tabela principal: Bit alterável = 0 (Valor falso)
   Tabela auxiliar: Bit alterável = 1 (Valor correto) e Bit alterado = 0

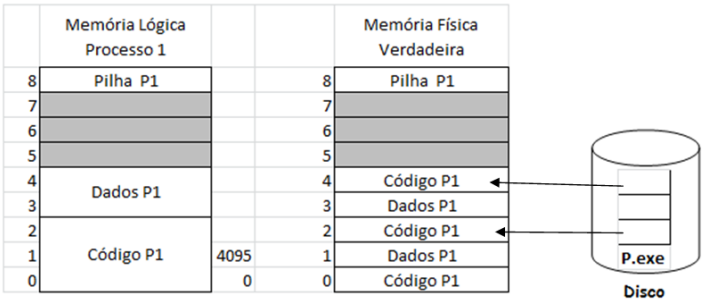
2) Quando o processo tenta alterar a pagina o HW gera a interrupção. O SO verifica se a pagina é realmente alterável na Tabela Auxiliar. Esta interrupção só será gerada quando o processo alterar a página.

3) Tabela auxiliar: Bit alterado = 0
   Tabela principal: Bit alterável = 1 (Valor correto)

MECANISMO UTILIZADOS NA OTIMIZAÇÃO PARA REDUZIR O USO DO DISCO NA PAGINAÇÃO SOB DEMANDA

CARREGAMENTO DAS PÁGINAS LOGICAS DE CÓDIGO "SOB DEMANDA"

O SO faz uma coisa diferente para evitar o uso do disco. Conforme mostra a Figura abaixo o processo tem um arquivo executável que está no disco. As paginas de código do processo não são todas carregadas na memoria física quando o processo começa. Logo as páginas de códigos não são válidas na Tabela de Páginas do processo, conforme mostra a Figura.



Mapa de Memória do Processo 1

Tipo	Pagina Inicial	Qtd Páginas
Codigo	0	3
Dados	3	2
Pilha	8	1

Como o código não está na memória física, quando este processo entra em execução e vai chamar o programa principal na primeira rotina do processo fará um salto para uma das paginas do código. Geralmente a pagina logica 0. Ela estará invalida. Então o HW gera uma interrupção. O SO vai tratar essa interrupção e não abortará o processo.

O que está na página logica 0 é diferente da página logica 6 conforme mostra a Figura da Tabela de Página do Processo. A página logica 0 pertence a área de código. A pagina logica 6 não, o SO sabe disso consultando a Tabela Mapa de Memória.

A interrupção foi gerada simplesmente porque o SO não carregou ainda a pagina logica 0. Não é uma pagina logica que está invalida de verdade ela existe só que não foi carregada para memoria ainda. No momento que o SO vai tratar essa interrupção ele coloca na memoria física a pagina logica 0 e sinaliza na Tabela de Pagina do Processo. Nesse somente um pagina do código esta na memoria as outras duas paginas estão no disco. E o processo é executado.

Tabela de Páginas do Processo 1

Num_pag_logica	Num_pag_Fisica	Válido	Executável	Alterável	Núcleo	Alterado (Dirty = sujo)
8						
7						
6						
5						
4						
3	5	1	0	1	0	0
2						
1						
0						

Se o processo chamar alguma sub-rotina que está na pagina logica do código que se encontra no disco a mesma coisa vai acontecer. Vai gerar uma interrupção e o SO ao tratar essa interrupção percebe que precisa traz para memória a pagina logica que está no disco. E preenche a Tabela de Pagina do Processo.

O SO carrega as paginas de código para memoria física aos poucos. Isto é feito quando o processo precisa utilizar a pagina pela primeira vez. Dai o nome de carregamento "sob demanda". O código só é carregado quando ele é demandado. Quando uma sub rotina precisa dele. É esse mecanismo que dá o nome a paginação com o uso do disco que é paginação sob demanda.

POOL DE PAGINAS VITIMAS

É adiantar o salvamento no disco das paginas vitimas. No momento que a memoria física está quase cheia. O SO pré escolhe as paginas vitimas. O SO a principio só escolhe uma pagina vitima quando a memoria física encheu. O que acontece é que muitas vezes tem que liberar espaço na memoria física pra trazer um pagina logica que está no disco.

O SO adianta esse salvamento no momento que o disco está sem uso. Assim não terá perda de desempenho, pois o disco não está sendo utilizado. Quando o processo precisar de memória não será preciso escolher e salvar uma pagina vitima em disco, pois isto já foi feito e tem memoria física livre.

Com este mecanismo a memoria física nunca fica cheia, pois quando está quase cheia o SO escolher uma pagina vitima e salva em disco quando o disco não está sendo utilizado. Esse mecanismo também é chamado de **pool de paginas vitimas**. O SO sempre mantem um numero de paginas vitimas para serem salvas em disco.

Portanto, o SO aproveita o tempo ocioso da CPU para procura paginas vitimas e salva-as em disco. Gerando um "pool" de páginas livres liberando espaço na memoria. Quando determinado percentual de paginas livres é alcançado à procura de paginas vitimas é iniciada.

Vantagens do POOL De Paginas Vitimas

-
-
-

Desvantagens do POOL De Paginas Vitimas

-
-
-

ALOCAÇÃO DE PÁGINAS FÍSICAS A UM PROCESSO

A alocação de páginas físicas a um processo é o relacionamento de paginas físicas e um processo. Existem duas formas de fazer a alocação de página físicas que são:

1) **Alocação Global** → Não há regra que controle quantas paginas físicas cada processo tem. Ex.: Unix.

2) **Alocação Local** → Há regra que define a quantidade mínima de paginas física que um processo deve ter. Ex.: Windows

Alocação Global

É escolhida como vítima a página de qualquer processo. Não existe número de páginas por processo. Assim, o processo pode receber páginas físicas de outros processos que foram escolhidas como vítima.

Exemplo: Um processo gerou uma interrupção por falta de pagina porque não tem memoria física, pois ela está cheia. O SO tem que escolher uma pagina vitima. Como não tem regra nenhuma o SO escolhe qualquer pagina física como vitima pode ser ou não do processo que gerou interrupção. Nesse caso será utilizado o algoritmo puro. Digamos que seja executado o LRU. A pagina mais antiga será a pagina vitima independente do processo, pois não tem regra nenhuma que indica de qual processo tem que ser a pagina vitima.

Vantagens da Alocação Global

-
-
-

Desvantagens da Alocação Global

- Um processo que ocupa muita memória roubará a memória de outros processos. Quando o processo que teve suas páginas físicas roubadas volta a executar gerará muitas faltas de paginas, pois todas suas paginas estão no disco. Isto deixa a execução do processo muito lenta.
-
-

Alocação Local

Cada processo tem um conjunto de paginas físicas reservadas. Uma página do próprio processo é escolhida como vítima para dar vez à outra página do próprio processo. Logo, se este processo gerar falta de pagina procura-se paginas vitimas aplicando um dos Algoritmos de Escolha de Página Vitima (FIFO, LRU, 2ª CHANCE) no conjunto de paginas físicas reservadas para o processo.

Na Alocação Local a memoria física fica cheia quando todas as paginas logicas reservadas estão sendo utilizadas.

A alocação local é vantajosa em relação à alocação global, mas é difícil de ser implementada. Sua maior dificuldade é definir a quantidade de paginas física para ser alocada a cada processo porque cada processo pode precisar de uma quantidade de memória física diferente.

Um processo que não alocou muita memoria logica tem uma área de código pequena, uma área de dados pequena e uma área de pilha pequena. Esse processo não vai utilizar muita memoria física, pois tem áreas pequenas. Um processo que tenha alocado muita memoria tem uma área de código, dados e pilha grande. Exemplo alocação de arrays de 100 MB.

Alocar memoria não significa usar memoria são coisas diferentes. O processo que está utilizando toda sua memoria precisa de mais memoria física para poder rodar. Não é muito obvio para o Sistema Operacional definir qual é a quantidade de paginas físicas que um processo precisa ter porque depende do processo.

A ideia é não ter uma quantidade fixa de memoria física reservada para cada processo, mas sim uma quantidade variável que depende de cada processo. Para conseguir definir a quantidade de paginas física a ser alocada para cada processo o SO utiliza alguns conceitos que são: **WORKING SET, PRINCIPIO DA LOCALIDADE E THRASHING.**

Vantagens da Alocação Local

- Evita que processos comedores de memória roubem memória de outros processos. Desvantagem: Como saber o conjunto de paginas que deve ser alocado.
-
-

Desvantagens da Alocação Local

-
-
-

WORKING SET

É o conjunto de paginas logicas que um processo utiliza durante certo intervalo de tempo, ou seja, é o conjunto de páginas lógicas necessárias para a execução de um processo.

Um processo que ocupa quase toda a memória no caso da alocação global pode pegar páginas físicas de outros processos. Muitos processos em execução podem causar falta de páginas.

Um processo costuma utilizar as mesmas paginas associadas à determinada rotina do processo. Isto minimiza o numero de faltas e só volta a gerar novas faltas quando o processo muda o Working Set até que todas as paginas do Working Set sejam carregadas na memória, então, o numero de faltas cairá. Quanto maior o numero de processos, maior a chance da CPU estar sendo utilizada.

O Windows NT tem um número mínimo e um número máximo de páginas, o processo não pode ter menos que o mínimo e mais que o máximo.

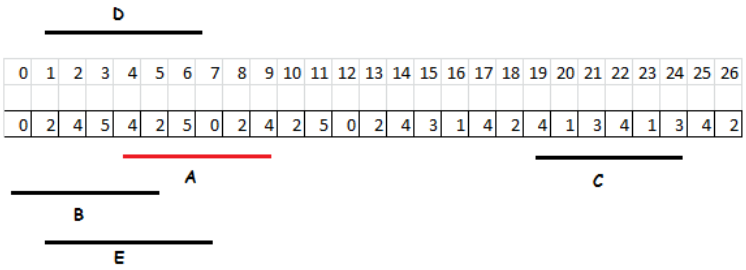
A dificuldade do algoritmo é para saber o número de páginas físicas que devem ser alocadas para cada processo. Este número tem que ser maior ou igual ao Working Set do processo. Se o SO conseguir alocar um número de páginas maior ou igual ao Working Set do processo, a falta de página quase não vai existir, vai existir falta de página quando trocar o Working Set do processo.

Se o número de páginas físicas for menor que o Working Set do processo, vai ocorrer falta de página o tempo todo, pois uma página que foi escolhida como vítima deverá ser usada logo em seguida, como não está na memória vai gerar falta de página.

Não tem como o SO saber o Working Set do processo, o que ele faz é observar o número de falta de página do processo, se esse número for grande, ele fica sabendo que o Working Set do processo é maior que o número de páginas físicas que o processo tem.

Para resolver isso, ele aumenta o número de páginas físicas desse processo. Com isso, enquanto esse Working Set não mudar e se já estiver todo na memória, não vai ocorrer falta de página para esse processo. A falta de página só ocorre quando trocar o Working Set. Por outro lado, se o SO observar que o processo tem mais páginas físicas que o Working Set, ele pode diminuir o número de páginas físicas alocadas ao processo.

Através do Working Set o SO sabe qual o número de páginas necessárias para cada processo. Por exemplo, A Figura abaixo mostra uma sequencia de paginas logicas acessadas por um processo:



WS(5) = {0,2,4,5}

WS(9) = {0,2,4,5}

WS(24) = {1,3,4}

WS(6) = {2,4,5} WS estável {0,2,4,5}

WS(7) = {0,2,4,5}

WS(14) = {0,2,4,5}

WS(15) = {0,2,3,4,5}

WS(16) = {0,1,2,3,4,5} Momento de transição

WS(17) = {0,1,2,3,4}

WS(18) = {1,2,3,4}

WS(19) = {1,2,3,4} WS estável {1,2,3,4}

WS(20) = {1,2,3,4}

(M) WS(21) = {1,2,3,4}

O Working Set é definir o que é um intervalo de tempo e vê quais paginas foram utilizadas nesse intervalo. No exemplo, está sendo utilizado o intervalo de tempo igual a 6.

Conforme podemos verificar na Figura ao lado e no WS analisado em alguns instantes de tempo até o tempo 14 o WS está entre {0,2,4,5} do 15 até o 17 o WS é instável. Do 18 em diante volta a ser estável {1,2,3,4} e se mantém estável por algum tempo. Tem um padrão para o uso do WS.

Se o número de páginas físicas reservadas para esse processo for maior ou igual ao Working Set não haverá falta de páginas.

Se o número de páginas físicas reservadas para esse processo for um pouco menor ao Working Set haverá pouca falta de páginas.

Se o número de páginas físicas reservadas para esse processo for muito menor ao Working Set haverá muita falta de páginas.

O SO acompanha o número de falta de página dos processos. Se ele perceber que o número de faltas de determinado processo é muito grande, ele pode reservar mais páginas físicas para esse processo.

Se acabar as páginas físicas, o SO diminui o número de páginas físicas alocadas para todos os processos. Se algum processo sofrer muita falta de páginas, o SO reservará mais páginas físicas para ele.

O Working Set pode alterar se for executada outra rotina. Haverá falta de página na mudança do Working Set.

Vantagens do Working Set

-
-
-

Desvantagens do Working Set

-
-
-

PRINCIPIO DA LOCALIDADE (LOCALIDADE DE ACESSO)

Os processos tendem a continuar usando o conjunto de paginas logicas que já estão usando. Em outras palavras o WS tende a ficar estável. Isto é chamado de Principio de localidade ou Localidade de acesso.

O WS é uma função matemática que leva em consideração o intervalo de observação e os acessos ao longo do tempo e se consegue atribuir o valor. O comportamento de estabilidade do WS é muito importante, pois com esse comportamento pode-se pensar em uma logica para definir qual é a quantidade de páginas físicas que cada processo deve ter.

Problema: Qual a quantidade de páginas físicas que deve ser reservada a cada processo?

Solução: Quantidade de paginas físicas reservadas = Tamanho do WS do processo

No exemplo de WS apresentado no tópico anterior, o tamanho do WS do processo é 4. Enquanto o WS for estável o processo não gera interrupção, pois todas paginas logicas que ele precisa estão na memoria física. Se ao contrário o SO tiver reservado para este processo apenas 2 paginas físicas e o tamanho do WS é 4, não ira satisfazer o processo já que gera interrupção impactando na execução do processo que será muito lenta.

Quando o WS muda gera interrupção. O WS tem 4 paginas reservadas para esse processo. No intervalo de tempo 17 tem um WS instável que usa outras paginas logicas que não estavam sendo utilizadas antes. No instante 15 passa a usar a pagina 3.

Então nesse momento o SO vai buscar no disco a pagina logica 3 e colocar na memoria física. Depois no instante 16 o processo passa a usar a pagina logica 1. Nesse momento terá um interrupção pela falta de pagina e o SO busca no disco a pagina logica 1. No instante 17 é solicita a pagina logica 4, mas ela já está na memoria física então não gera interrupção.

Na transição do WS temos algumas faltas de paginas, mas no momento seguinte já não tem mais, pois se estabilizou novamente.

Se esta logica fosse admitida pelo SO teríamos faltas de paginas somente na transição do WS. Nessa transição não teríamos muitas faltas de paginas. Então essa é uma boa logica, pois otimiza a quantidade de paginas físicas reservadas sem sobrar muito e minimiza a quantidade de falta de paginas.

Outra solução para falta de paginas é aumentar em muito o numero de paginas por processo. O processo não terá falta de paginas, mas o processo terá muito mais memoria física do que precisa de verdade sendo um desperdício de memoria física. Por exemplo, se der 20 pagina físicas para esse processo que só utiliza 4 temos uma diminuição na falta de pagina, no entanto, aumento no gasto da memoria.

A ideia do Principio de localidade é manter Qtd de paginas físicas reservadas = Tamanho do WS.

O problema é que isto é difícil de ser implementado, pois o SO não sabe a sequencia exata de paginas que o processo acessou. Se ele soubesse conseguiria calcular o Tamanho do WS de cada processo. Logo, na pratica essa solução não resolve o problema.

Embora o SO não saiba o WS exato de cada processo com esta logica montada ele consegue inferir se a igualdade está ocorrendo ou não. A tabela abaixo lista os casos que o SO consegue ou não resolver o problema da Quantidade de paginas físicas reservadas:

Caso	Descrição	Consequência (Efeito)	Ação do SO
1	Qtd de Pag Fís reservadas = Tamanho do WS	Poucas faltas de paginas (ocorrem na transição entre WS estáveis).	O SO não consegue fazer nada nesta situação.
2	Qtd de Pag Fís reservadas < Tamanho do WS	Muitas faltas de páginas.	Nesse caso o SO consegue induzir o problema e resolver. Ele aumenta a Qtd de Pag Fis reservadas.
3	Qtd de Pag Fís reservadas > Tamanho do WS	Poucas faltas de paginas.	O SO não consegue fazer nada nesta situação.

No caso 2 o SO consegue chegar a seguinte conclusão que a reserva de paginas físicas não está igual ao Tamanho do WS. Então ele aumenta o tamanho da reserva de paginas físicas. Continua gerando muitas faltas de paginas aumenta mais o tamanho da reserva. Quando passar a gerar pouca falta de paginas é porque possivelmente se chegou à igualdade.

O caso 3 pode ser uma consequência do caso 2. Mas, o SO não tem como saber se o processo tem mais paginas físicas reservadas do que precisa.

Conforme os processos são colocados em execução o SO reserva paginas físicas para esses processos. E aumenta essa reserva pelo mecanismo do caso 2 explicado acima. Pode chegar um momento que um novo processo é colocado em execução e o SO queira reservar paginas físicas para este processo e não tenha paginas física reservável porque todas as paginas estão reservadas para os processos anteriores a este processo.

Nesse caso o SO tem um problema. Ele pode não deixara que esse novo processo seja executado. Essa é uma solução, pois não tem como reservar pagina física. Mas essa solução pode ser ruim, pois pode ter processos que estão com uma reserva de pagina física exagerada.

Sendo assim, não executar mais um processo seria desperdício porque se não existisse processos com reservas exagerada o HW conseguiria suportar mais um processo. Para que isto não aconteça o SO supõem que existem casos de reservas exageradas e diminui a reserva de paginas físicas de todos os processos.

Ao diminuir a reserva o SO observa a quantidade de falta de paginas dos processos. Se um processo tiver pouca falta de pagina o SO sabe que a reserva de paginas desse processo está correta, mas se um processo tiver muitas faltas de paginas o SO sabe que diminui muito a reserva de paginas desse processo e tem que voltar a quantidade original de reserva que esse processo tinha.

O SO guarda a quantidade de paginas (WS) de todos os processos antes de diminuir a quantidade de paginas do processo para poder alocar um novo processo quando não tem reserva de paginas físicas.

Mesmo com deficiências essa regra é boa, pois é melhor do que nada, pois é preciso ter alguma regra para reservar paginas físicas. O Windows utiliza essa Regra. O Unix não tem regra para reserva de paginas físicas. Quando não se tem regra para reserva de paginas um único processo pode roubar a reserva de paginas físicas dos outros processos. Isto é ruim, pois quem usa muito recebe muito e quem usa pouco pode ate ficar sem reserva.

No entanto, no Unix um processo desse tipo não é frequente. Então e raro acontecer de um processo ficar sem reserva de pagina e outro processo ficar com uma reserva muito grande de paginas físicas. No Windows essa situação de roubo total nunca ocorrerá, pois a reserva é garantida para todos os processos usando muita ou pouca paginas física.

Vantagens do Principio Da Localidade

-
-
-

Desvantagens do Principio Da Localidade

-
-
-

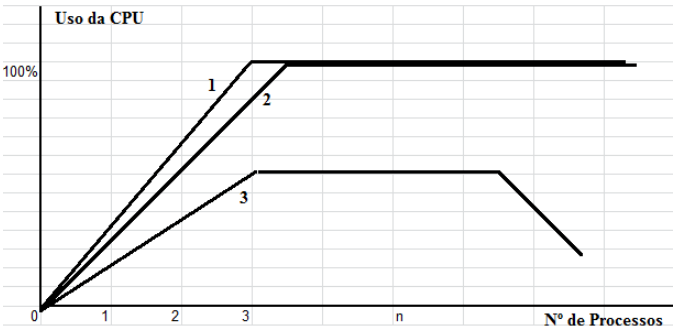
<u>THRASHING (USO DA CPU)</u>

O Uso da CPU é determinado pela quantidade de vezes que ela executou código útil durante um instante de tempo. Um processo que faz um cálculo demorado pode levar o Uso da CPU a 100%. Não é comum ter processos este tipos de processo, pois tipicamente o processo executa alguma coisa e em seguida espera alguma coisa acontecer (fica bloqueado).

Quantos mais processos típicos existirem na memoria para serem executados mais o uso da CPU irá aumentar. Vai chegar uma hora que o uso da CPU será de 100%, ou seja, a CPU está executando código útil o tempo todo.

A paginação sob demanda introduz bloqueios na execução de um processo que em uma maquina sem paginação não tem. Logo a CPU é menos usada em um maquina com paginação. Em uma maquina com paginação quando a memoria enche não dá erro para o usuário já em um maquina sem paginação quando a memoria fica cheia dá erro para o usuário.

O problema é que quando não se tem uma gerencia de memoria sofisticada pode ter situações que tem tantos processos na memoria física que ela não suporta todos esses processos. Então na maior parte do tempo esses processos ficaram bloqueados esperando as paginas virem do disco, pois como não tem memoria física suficiente as paginas são salvas no disco. O numero de bloqueios passa a ser tão grande que o desempenho da maquina (uso de CPU) diminui ao invés de aumentar. Essa situação é chamada de **Thrashing** (Arrastamento) - nome dado à lentidão da máquina causada pelo excesso de falta de página.



(1) Máquina sem paginação
(2) Máquina com paginação sob demanda
(3) Thrashing

O Thrashing é comum ocorrer em maquinas com Alocação Global. No caso da Alocação Local isto é muito menos provável de ocorrer. Pois, como cada processo tem uma reserva de paginas física para garantir uma boa execução do processo sem gerar muita falta de paginas. Logo, se a Alocação Local for bem implementada não gerará a situação de Thrashing.

Windows não tem garantia que a Alocação Local não terá Thrashing, mas é mais frequente acontecer no Unix. Hoje em dia, o problema Thrashing não é muito importante, pois as memorias físicas são bem grandes.

O Windows 95 tinha somente Alocação Global então o Thrashing ocorria com mais frequência do que no Windows atual que tem Alocação Local. O efeito do Thrashing para o usuário é lentidão e o Sistema mostra que Uso da CPU está baixo.

Solução do Unix para resolver o Thrashing é o Swap de processos. No Unix quando o SO percebe que esta ocorrendo muitas faltas de paginas de forma geral ele aciona o mecanismo de Swap de Processo, ou seja, quando a memoria acaba o SO escolhe um processo e salva todo ele no disco. Ao fazer isso à competição pela CPU é reduzida logo a competição pela memoria diminui.

As causas para ocorrência do Thrashing são:

- ✓ Processo comedor de memória, ou seja, quando um processo que necessita de muita memória;
- ✓ Muitos processos em execução, ou seja, se tiver muitos processos que alocam e usam a memoria física acaba acontecendo de ocupar toda a memória da máquina.

Portanto, a falta de página provoca o Thrashing porque bloqueia o processo. Poucas faltas não afetam muito, mas se as faltas forem muitas podem provocar pouco Uso da CPU. 1% de falta de página já é um número muito grande e o suficiente para acontecer isso.

Vantagens do Thrashing

- ➢
- ➢
- ➢

Desvantagens do Thrashing

- ➢
- ➢
- ➢

Vantagens da Paginação Sob Demanda

- ➢ É mais rápida. E se uma pagina do processo for salva em disco impedi do processo continuar executando. No SWAP de processo quando um processo é salvo em disco muda o estado dele. No SWAP de paginas ao salvar uma pagina no disco não muda o estado do processo. O processo só não entrara em execução se tentar executar uma pagina que foi salva em disco. Mas, enquanto executar paginas validas funciona normalmente. Esta é uma vantagem importante do SWAP de paginas.;
- ➢ Outra vantagem é em relação ao tamanho. Na paginação como todas paginas tem o mesmo tamanho se precisou liberar espaço na memória física pode escolher qualquer um pagina física para ser salva em disco que está bom, pois tem o mesmo tamanho da pagina logica.

- No SWAP de processos se um usuário quer usar em certo processo que está em disco. O processo tem um tamanho e não é necessariamente igual a outro processo na memória. Então as vezes para trazer um processo para memoria é necessário talvez salvar três processos em disco.

- Quando uma pagina de código não é carregada para o disco o processo inicia sua execução mais rapidamente. Isto acontece com o Word o usuário consegue começar a utilizar o programa mais rápido.

- Ao carregar um código sob demanda se gasta menos memória física.

Desvantagens da Paginação Sob Demanda

- Introdução de bloqueios no processo em lugares que não se esperava, por exemplo, quando o processo é salvo no disco.

- Não se aplica em processos de tempo real, pois esses tipos de processos que tem a necessidade de executar muito rápido, por exemplo, uma maquina ou um robô. Se ocorrer uma parada de repente e robô estava andando ele pode andar mais do que deveria. Em um processo cujo tempo é critico não é recomendável que você use paginação com uso de disco. Porque ele introduz paradas (bloqueios) em momentos que você não esperaria que acontecesse. Este é o problema da paginação em disco.

-

Observação

- Se uma mesma página é usada por dois ou mais processos e a mesma é transferida para o disco, as tabelas de páginas dos processos que a usam precisam ser atualizadas mudando o bit de validade dessa página de 1 para 0.

- A TLB existe sempre que existir paginação. Ela é controlada pelo HW e não pelo SO.O SO escolheu a pagina logica 7 como vitimas. Ele tem que mudar o bit valido da pagina logica e atualizar a TLB. Existe uma instrução para controlar a TLB quando uma pagina logica é levada para o disco. O SO avisa a TLB através de uma instrução que avisa o seguinte se estiver um registro para pagina logica 7 na TLB pode limpar. A instrução que o SO manda não sabe se existe ou não a pagina logica que está sendo salva no disco na TLB. Se a pagina logica estiver na TLB o HW vai retirá-la.

- Além dos 4 mecanismos para solução existe o caso de se Criar um arquivão de paginas logicas salvas em disco chamado de arquivo de SWAP. Este arquivo pode ficar em um arquivo grande ou em uma partição separada. Somente o SO tem acesso a este arquivo quem tenta mexer dá erro e não consegue mexer. Assim, o SO consegue saber quais são as paginas salvas em disco e não salva esta pagina novamente.
- .

- A Microsoft mistura os conceitos de alocação local e working set.

- O thrashing é normal ocorrer em SO que utilizam alocaçãp global, por exemplo, linux

SEGMENTAÇÃO VERSUS PAGI NAÇÃO

Qual dos esquemas é o melhor? Talvez não exista resposta definitiva para esta questão, pois se trata de um assunto polêmico que vem sendo discutido há muito tempo. Contudo, pode-se apresentar as vantagens (e desvantagens) de cada um. A seguir são apresentadas as principais diferenças entre segmentação e paginação.

Fragmentação: Com a segmentação existe fragmentação externa, com paginação existe fragmentação interna (em média, perda de ½ página por processo). Aqui a paginação parece ser vantajosa (desde que os tamanhos das páginas não sejam muito grandes).

Administração da memória: é muito mais simples na paginação, já que qualquer página física pode ser usada para carregar qualquer página lógica. Praticamente, basta o SO manter uma lista com os números das páginas físicas disponíveis, para controlar a alocação de memória.

Proteção e compartilhamento: Aqui a segmentação é bem melhor, pois os segmentos constituem as unidades lógicas do programa (isto é, cada segmento tem um determinado significado ou função). Cada segmento poderá ser compartilhado e ter associada a si uma determinada proteção. Com paginação as partes de um programa estão aglutinadas, formando um bloco único (em uma mesma página podem estar funções, subrotinas e dados, por exemplo).

Espaço de endereçamento: Com paginação o espaço de endereçamento lógico é um espaço único (unidimensional), contínuo, com endereços que vão desde zero até MAX (onde MAX é o tamanho do programa menos 1).

Com segmentação o espaço lógico é formado por um conjunto de segmentos, cada segmento u com endereços que vão de zero até MAXu (onde MAXu é o tamanho de u menos 1).

Na paginação o espaço lógico é contínuo, pois se pegarmos o último endereço dentro da página p e somarmos 1, teremos o primeiro endereço da página p+1. Por exemplo, considerando que (p)2 seja a representação binária de p, tem-se que o último endereço da página p é "(p)2111...1". Somando-se 1 tem-se "(p+1)2000...0", que é o primeiro endereço da página p+1.

Isto não ocorre na segmentação, onde o espaço lógico é descontínuo. Aqui não se pode dizer que haja vantagem de um esquema sobre o outro. As vantagens e desvantagens estão nas consequências destas organizações.