# Gerenciamento de memória

A parte do sistema operacional que gerencia (parte da) hierarquia de memórias é chamada de gerenciador de memória. Sua função é gerenciar eficientemente a memória: controlar quais partes estão sendo usadas, alocar memória para processos quando eles precisam dela e liberá-la quando terminarem.

## Sem abstração de memória

A abstração de memória mais simples é não ter abstração alguma. Até 1980 os computadores não tinham abstração de memória. Cada programa apenas via a memória física, mesmo assim, várias opções eram possíveis:

1 - O sistema operacional pode estar na parte inferior da memória em RAM;
2 - No topo da memória
3- Os drivers do dispositivo talvez estejam no topo da memória em um ROM e o resto do sistema em RAM bem abaixo

Nessas condições, não era possível ter dois programas em execução na memória ao mesmo tempo. Se o primeiro programa escrevesse um novo valor para, digamos, a posição 2000, esse valor apagaria qualquer valor que o segundo programa estivesse armazenando ali. Nada funcionaria e ambos os programas entrariam em colapso quase que imediatamente.

O primeiro modelo foi usado antes em computadores de grande porte e minicomputadores, mas raramente é usado. O segundo modelo é usado em alguns computadores portáteis e sistemas embarcados. O terceiro modelo foi usado pelos primeiros computadores pessoais (por exemplo, executando o MS-DOS), onde a porção do sistema no ROM é chamada de BIOS.

Uma maneira de se conseguir algum paralelismo em um sistema sem abstração de memória é programá-lo com múltiplos threads. Como todos os threads em um processo devem ver a mesma imagem da memória, o fato de eles serem forçados a fazê-lo não é um problema. Embora essa ideia funcione, ela é de uso limitado, pois o que muitas vezes as pessoas querem é que programas não relacionados estejam executando ao mesmo tempo, algo que a abstração de threads não realiza. Além disso, qualquer sistema que seja tão primitivo a ponto de não proporcionar qualquer abstração de memória é improvável que proporcione uma abstração de threads.

Mesmo sem uma abstração de memória, é possível executar múltiplos programas ao mesmo tempo. O que um sistema operacional precisa fazer é salvar o conteúdo inteiro da memória em um arquivo de disco, então introduzir e executar o programa seguinte.

Embora o endereçamento direto de memória física seja apenas uma memória distante nos computadores de grande porte, minicomputadores, computadores de mesa, notebooks e smartphones, a falta de uma abstração de memória ainda é comum em sistemas embarcados e de cartões inteligentes.

Casos em que o software se endereça à memória absoluta: rádios, máquinas de lavar roupas e fornos de micro-ondas.

Smartphones, por exemplo, possuem sistemas operacionais elaborados.

## Realocação

O problema fundamental aqui é que ambos os programas referenciam a memória física absoluta, e não é isso que queremos, de forma alguma. O que queremos é cada programa possa referenciar um conjunto privado de endereços local a ele. Mostraremos como isso pode ser conseguido. O que o IBM 360 utilizou como solução temporária foi modificar o segundo programa dinamicamente enquanto o carregava na memória, usando uma técnica conhecida como realocação estática. Ela funcionava da seguinte forma: quando um programa estava carregado no endereço 16.384, a constante 16.384 era acrescentada a cada endereço de programa durante o processo de carregamento (de maneira que “JMP 28” tornou-se “JMP 16.412” etc.). Conquanto esse mecanismo funcione se feito de maneira correta, ele não é uma solução muito geral e torna lento o carregamento. Além disso, exige informações adicionais em todos os programas executáveis cujas palavras contenham ou não endereços (realocáveis).

## Espaços de endereçamento

Dois problemas têm de ser solucionados para permitir que múltiplas aplicações estejam na memória ao mesmo tempo sem interferir umas com as outras: proteção e realocação.

Um espaço de endereçamento é o conjunto de endereços que um processo pode usar para endereçar a memória. Cada processo tem seu próprio espaço de endereçamento, independente daqueles pertencentes a outros processos (exceto em algumas circunstâncias especiais onde os processos querem compartilhar seus espaços de endereçamento).

Exemplos:
- números de telefones;
- portas de E/S;
- endereços de IPv4;
- conjunto de domínios da internet .com.

## Registradores base e registradores limite

Quando esses registradores são usados, os programas são carregados em posições de memória consecutivas sempre que haja espaço e sem realocação durante o carregamento.

## Troca de processos (Swapping)

Estratégia simples para lidar com a sobrecarga de memória que consiste em trazer cada processo em sua totalidade, executá-lo por um tempo e então colocá-lo de volta no disco.

Processos ociosos estão armazenados em disco em sua maior parte, portanto não ocupam qualquer memória quando não estão sendo executados.

Quando as trocas de processos criam múltiplos espaços na memória, é possível combiná-los em um grande espaço movendo todos os processos para baixo, o máximo possível. Essa técnica é conhecida como compactação de memória.

## Quanta memória deve ser alocada para um processo quando ele é criado ou trocado?

Se os processos são criados com um tamanho fixo que nunca muda, então a alocação é simples: o sistema operacional aloca exatamente o que é necessário, nem mais nem menos. No entanto, um problema ocorre sempre que um processo tenta crescer.

Se houver um espaço adjacente ao processo, ele poderá ser alocado e o processo será autorizado a crescer naquele espaço.

Por outro lado, se o processo for adjacente a outro, aquele que cresce terá de ser movido para um espaço na memória grande o suficiente para ele, ou um ou mais processos terão de ser trocados para criar um espaço grande o suficiente.

## Quanta memória deve ser alocada para um processo quando ele é criado ou trocado?

Se um processo não puder crescer em memória e a área de troca no disco estiver cheia, ele terá de ser suspenso até que algum espaço seja liberado (ou ele pode ser morto).

Se o esperado for que a maioria dos processos cresça à medida que são executados, provavelmente seja uma boa ideia alocar um pouco de memória extra sempre que um processo for trocado ou movido, para reduzir a sobrecarga associada com a troca e movimentação dos processos que não cabem mais em sua memória alocada.

# Gerenciamento de memória com mapas de bits

Quando a memória é designada dinamicamente, o sistema operacional deve gerenciá-la. Em termos gerais, há duas maneiras de se rastrear o uso de memória: mapas de bits e listas livres.

Correspondendo a cada unidade de alocação há um bit no mapa de bits, que é 0 se a unidade estiver livre e 1 se ela estiver ocupada (ou vice-versa). Se um processo não puder crescer em memória e a área de troca no disco estiver cheia, ele terá de ser suspenso até que algum espaço seja liberado (ou ele pode ser morto).

O tamanho da unidade de alocação é uma importante questão de projeto. Quanto menor a unidade de alocação, maior o mapa de bits.

Um mapa de bits proporciona uma maneira simples de controlar as palavras na memória em uma quantidade fixa dela, porque seu tamanho depende somente dos tamanhos da memória e da unidade de alocação. O principal problema é que, quando fica decidido carregar um processo com tamanho de k unidades, o gerenciador de memória deve procurar o mapa de bits para encontrar uma sequência de k bits 0 consecutivos. Procurar em um mapa de bits por uma sequência de um comprimento determinado é uma operação lenta (pois a sequência pode ultrapassar limites de palavras no mapa); este é um argumento contrário aos mapas de bits.

# Gerenciamento de memória com listas encadeadas

Outra maneira de controlar o uso da memória é manter uma lista encadeada de espaços livres e de segmentos de memória alocados, onde um segmento contém um processo ou é um espaço vazio entre dois processos.

**first fit**: O algoritmo mais simples é first fit (primeiro encaixe). O gerenciador de memória examina a lista de segmentos até encontrar um espaço livre que seja grande o suficiente. O espaço livre é então dividido em duas partes, uma para o processo e outra para a memória não utilizada, exceto no caso estatisticamente improvável de um encaixe exato. First fit é um algoritmo rápido, pois ele procura fazer a menor busca possível.

**next fit**: Uma pequena variação do first fit é o next fit. Ele funciona da mesma maneira que o first fit, exceto por memorizar a posição que se encontra um espaço livre adequado sempre que o encontra. Da vez seguinte que for chamado para encontrar um espaço livre, ele começa procurando na lista do ponto onde havia parado, em vez de sempre do princípio, como faz o first fit. Simulações realizadas por Bays (1977) mostram que o next fit tem um desempenho ligeiramente pior do que o do first fit.

**best fit**: Outro algoritmo bem conhecido e amplamente usado é o best fit. O best fit faz uma busca em toda a lista, do início ao fim, e escolhe o menor espaço livre que seja adequado. Em vez de escolher um espaço livre grande demais que talvez seja necessário mais tarde,

o best fit tenta encontrar um que seja de um tamanho próximo do tamanho real necessário, para casar da melhor maneira possível a solicitação com os segmentos disponíveis.

O best fit é mais lento do que o first fit, pois ele tem de procurar na lista inteira toda vez que é chamado. De uma maneira um tanto surpreendente, ele também resulta em um desperdício maior de memória do que o first fit ou next fit, pois tende a preencher a memória com segmentos minúsculos e inúteis. O first fit gera espaços livres maiores em média.

**worst fit:** Para contornar o problema de quebrar um espaço livre em um processo e um trecho livre minúsculo, a solução poderia ser o worst fit, isto é, sempre escolher o maior espaço livre, de maneira que o novo segmento livre gerado seja grande o bastante para ser útil. No entanto, simulações demonstraram que o worst fit tampouco é uma grande ideia.

**quick fit:** Outro algoritmo de alocação é o quick fit, que mantém listas em separado para alguns dos tamanhos mais comuns solicitados. Por exemplo, ele pode ter uma tabela com n entradas, na qual a primeira é um ponteiro para o início de uma lista espaços livres de 4 KB, a segunda é um ponteiro para uma lista de espaços livres de 8 KB, a terceira de 12 KB e assim por diante. Espaços livres de, digamos, 21 KB, poderiam ser colocados na lista de 20 KB ou em uma lista de espaços livres de tamanhos especiais.

Com o quick fit, encontrar um espaço livre do tamanho exigido é algo extremamente rápido, mas tem as mesmas desvantagens de todos os esquemas que ordenam por tamanho do espaço livre, a saber, quando um processo termina sua execução ou é transferido da memória, descobrir seus vizinhos para ver se uma fusão com eles é possível é algo bastante caro. Se a fusão não for feita, a memória logo se fragmentará em um grande número de pequenos segmentos livres nos quais nenhum processo se encaixará.

# Memória virtual

Apesar de os tamanhos das memórias aumentarem depressa, os tamanhos dos softwares estão crescendo muito mais rapidamente.

Como consequência desses desenvolvimentos, há uma necessidade de executar programas que são grandes demais para se encaixar na memória e há certamente uma necessidade de ter sistemas que possam dar suporte a múltiplos programas executando em simultâneo, cada um deles encaixando-se na memória, mas com todos coletivamente excedendo-a.

Uma solução adotada nos anos 1960 foi dividir os programas em módulos pequenos, chamados de sobreposições. Quando um programa inicializava, tudo o que era carregado na memória era o gerenciador de sobreposições, que imediatamente carregava e executava a sobreposição 0. Quando terminava, ele dizia ao gerenciador de sobreposições para carregar a sobreposição 1, acima da sobreposição 0 na memória (se houvesse espaço para isso), ou em cima da sobreposição 0 (se não houvesse).

Método encontrado para passar todo o programa para o computador. A ideia básica é que cada programa tem seu próprio espaço de endereçamento, o qual é dividido em blocos chamados de páginas. De certa maneira, é uma generalização da ideia do registrador base e registrador limite.

Funciona bem em um sistema de multiprogramação, com pedaços e partes de muitos programas na memória simultaneamente. Enquanto um programa está esperando que partes de si mesmo sejam lidas, a CPU pode ser dada para outro processo.

# Paginação

Quando um programa executa uma instrução como MOV REG,1000 ele o faz para copiar o conteúdo do endereço de memória 1000 para REG. Esses endereços gerados por computadores são chamados de endereços virtuais e formam o espaço de endereçamento virtual.

Em computadores sem memória virtual, o endereço virtual é colocado diretamente no barramento de memória e faz que a palavra de memória física com o mesmo endereço seja lida ou escrita. Quando a memória virtual é usada, os endereços virtuais não vão diretamente para o barramento da memória. Em vez disso, eles vão para uma MMU (Memory Management Unit — unidade de gerenciamento de memória) que mapeia os endereços virtuais em endereços de memória física.

Técnica usada pela maioria dos sistemas de memória virtual. O espaço de endereçamento virtual consiste em unidades de tamanho fixo chamadas de páginas.

As unidades correspondentes na memória física são chamadas de quadros de página.

Transferências entre a memória RAM e o disco são sempre em páginas inteiras.

Muitos processadores dão suporte a múltiplos tamanhos de páginas que podem ser combinados e casados como o sistema operacional preferir.

No hardware real, um bit Presente/ausente controla quais páginas estão fisicamente presentes na memória.

A interrupção é chamada de falta de página (page fault). O sistema operacional escolhe um quadro de página pouco usado e escreve seu conteúdo de volta para o disco (se já não estiver ali). Ele então carrega (também do disco) a página recém referenciada no quadro de página recém-liberado, muda o mapa e reinicia a instrução que causou a interrupção.

O número da página é usado como um índice para a tabela de páginas, resultando no número do quadro de página correspondente àquela página virtual.

# Tabelas de páginas

O objetivo da tabela de páginas é mapear as páginas virtuais em quadros de páginas. Matematicamente falando, é uma função com o número da página virtual como argumento e o número do quadro físico como resultado. Usando o resultado dessa função, o campo da página virtual em um endereço virtual pode ser substituído por um campo de quadro de página, desse modo formando um endereço de memória física.

## Estrutura de uma entrada da tabela de páginas

O tamanho varia de computador para computador, mas 32 bits é um tamanho comum. O campo mais importante é o Número do quadro de página. Afinal, a meta do mapeamento de páginas é localizar esse valor. Próximo a ele, temos o bit Presente/ausente. Se esse bit for 1, a entrada é válida e pode ser usada. Se ele for 0, a página virtual à qual a entrada pertence não está atualmente na memória. Acessar uma entrada da tabela de páginas com esse bit em 0 causa uma falta de página.

Os bits Proteção dizem quais tipos de acesso são permitidos. Na forma mais simples, esse campo contém 1 bit, com 0 para ler/escrever e 1 para ler somente. Um arranjo mais sofisticado é ter 3 bits, para habilitar a leitura, escrita e execução da página.

Os bits Modificada e Referenciada controlam o uso da página. Ao escrever na página, o hardware automaticamente configura o bit Modificada. Esse bit é importante quando o sistema operacional decide recuperar um quadro de página. Se a página dentro do quadro foi modificada (isto é, está “suja”), ela também deve ser atualizada no disco. Se ela não foi modificada (isto é, está “limpa”), ela pode ser abandonada, tendo em vista que a cópia em disco ainda é válida. O bit às vezes é chamado de bit sujo, já que ele reflete o estado da página.

O bit Referenciada é configurado sempre que uma página é referenciada, seja para leitura ou para escrita. Seu valor é usado para ajudar o sistema operacional a escolher uma página a ser substituída quando uma falta de página ocorrer. Páginas que não estão sendo usadas são candidatas muito melhores do que as páginas que estão sendo, e esse bit desempenha um papel importante em vários dos algoritmos de substituição de páginas.

Por fim, o último bit permite que o mecanismo de cache seja desabilitado para a página. Essa propriedade é importante para páginas que mapeiam em registradores de dispositivos em vez da memória. Se o sistema operacional está parado em um laço estreito esperando que algum dispositivo de E/S responda a um comando que lhe foi dado, é fundamental que o hardware continue buscando a palavra do dispositivo, e não use uma cópia antiga da cache. Com esse bit, o mecanismo da cache pode ser desabilitado. Máquinas com espaços para E/S separados e que não usam E/S mapeada em memória não precisam desse bit.

A tabela de páginas armazena apenas aquelas informações de que o hardware precisa para traduzir um endereço virtual para um endereço físico.

A memória virtual pode ser implementada dividindo o espaço do endereço virtual em páginas e mapeando cada uma delas em algum quadro de página da memória física ou não as mapeando (temporariamente). Desse modo, ela diz respeito basicamente à abstração criada pelo sistema operacional e como essa abstração é gerenciada.

## Acelerando a paginação

Em qualquer sistema de paginação, duas questões fundamentais precisam ser abordadas:

- O mapeamento do endereço virtual para o endereço físico precisa ser rápido.
- Se o espaço do endereço virtual for grande, a tabela de páginas será grande

O projeto mais simples (pelo menos conceitualmente) é ter uma única tabela de página consistindo de uma série de registradores de hardware rápidos, com uma entrada para cada página virtual, indexada pelo número da página virtual.

As vantagens desse método são que ele é direto e não exige referências de memória durante o mapeamento. Uma desvantagem é que ele é terrivelmente caro se a tabela de páginas for grande; ele simplesmente não é prático na maioria das vezes. Outra desvantagem é que ter de carregar a tabela de páginas inteira em cada troca de contexto mataria completamente o desempenho.

## TLB ou memória Associativa

Considere, por exemplo, uma instrução de 1 byte que copia um registrador para outro. Na ausência da paginação, essa instrução faz apenas uma referência de memória, para buscar a instrução. Com a paginação, pelo menos uma referência de memória adicional será necessária, a fim de acessar a tabela de páginas. Dado que a velocidade de execução é geralmente limitada pela taxa na qual a CPU pode retirar instruções e dados da memória, ter de fazer duas referências de memória por cada uma reduz o desempenho pela metade. Sob essas condições, ninguém usaria a paginação.

Projetistas de computadores sabem desse problema há anos e chegaram a uma solução. Ela se baseia na observação de que a maioria dos programas tende a fazer um grande número de referências a um pequeno número de páginas, e não o contrário. Assim, apenas uma pequena fração das entradas da tabela de páginas é intensamente lida; o resto mal é usado.

Pequeno dispositivo de hardware para mapear endereços virtuais para endereços físicos sem ter de passar pela tabela de páginas.

Cada entrada contém informações sobre uma página, incluindo o número da página virtual, um bit que é configurado quando a página é modificada, o código de proteção

(ler/escrever/permissões de execução) e o quadro de página física na qual a página está localizada. Esses campos têm uma correspondência de um para um com os campos na tabela de páginas, exceto pelo número da página virtual, que não é necessário na tabela de páginas. Outro bit indica se a entrada é válida (isto é, em uso) ou não.

Quando um endereço virtual é apresentado para a MMU para tradução, o hardware primeiro confere para ver se o seu número de página virtual está presente na TLB comparando-o com todas as entradas simultaneamente (isto é, em paralelo). É necessário um hardware especial para realizar isso, que todas as MMUs com TLBs têm. Se uma correspondência válida é encontrada e o acesso não viola os bits de proteção, o quadro da página é tirado diretamente da TLB, sem ir à tabela de páginas. Se o número da página virtual estiver presente na TLB, mas a instrução estiver tentando escrever em uma página somente de leitura, uma falha de proteção é gerada.

O interessante é o que acontece quando o número da página virtual não está na TLB. A MMU detecta a ausência e realiza uma busca na tabela de páginas comum. Ela então destitui uma das entradas da TLB e a substitui pela entrada de tabela de páginas que acabou de ser buscada. Portanto, se a mesma página é usada novamente em seguida, da segunda vez ela resultará em uma presença de página em vez de uma ausência. Quando uma entrada é retirada da TLB, o bit modificado é copiado de volta na entrada correspondente da tabela de páginas na memória. Os outros valores já estão ali, exceto o bit de referência. Quando a TLB é carregada da tabela de páginas, todos os campos são trazidos da memória.

## Gerenciamento da TLB por software

Quando ocorre uma ausência de TLB, em vez de a MMU ir às tabelas de páginas para encontrar e buscar a referência de página necessária, ela apenas gera uma falha de TLB e joga o problema no colo do sistema operacional.

Para reduzir as ausências de TLB, às vezes o sistema operacional pode usar sua intuição para descobrir quais páginas têm mais chance de serem usadas em seguida e para pré-carregar entradas para elas na TLB. Por exemplo, quando um processo cliente envia uma mensagem a um processo servidor na mesma máquina, é muito provável que o processo servidor terá de ser executado logo. Sabendo disso, enquanto processa a interrupção para realizar o send, o sistema também pode conferir para ver onde o código, os dados e as páginas da pilha do servidor estão e mapeá-los antes que tenham uma chance de causar falhas na TLB.

A maneira normal para processar uma ausência de TLB, seja em hardware ou em software, é ir até a tabela de páginas e realizar as operações de indexação para localizar a página referenciada. O problema em realizar essa busca em software é que as páginas que armazenam a tabela de páginas podem não estar na TLB, o que causará faltas de TLB adicionais durante o processamento. Essas faltas podem ser reduzidas mantendo uma cache de software grande (por exemplo, 4 KB) de entradas em uma localização fixa cuja

página seja sempre mantida na TLB. Ao conferir a primeira cache do software, o sistema operacional pode reduzir substancialmente as ausências de TLB.

**Ausência leve (soft miss)**: ocorre quando a página referenciada não se encontra na TLB, mas está na memória. Tudo o que é necessário aqui é que a TLB seja atualizada.

**Ausência completa (hard miss)**: ocorre quando a página em si não está na memória (e, é claro, também não está na TLB).

**Passeio na tabela de páginas (page table walk)**: procurar o mapeamento na hierarquia da tabela de páginas.

Uma ausência não é somente leve ou completa. Algumas ausências são ligeiramente leves (ou mais completas) do que outras. Por exemplo, suponha que o passeio de página não encontre a página na tabela de páginas do processo e o programa incorra, portanto, em uma falta de página. Há três possibilidades. Primeiro, a página pode estar na realidade na memória, mas não na tabela de páginas do processo. Por exemplo, a página pode ter sido trazida do disco por outro processo. Nesse caso, não precisamos acessar o disco novamente, mas basta mapear a página de maneira apropriada nas tabelas de páginas. Essa é uma ausência bastante leve chamada falta de página menor (minor page fault). Segundo, uma falta de página maior (major page fault) ocorre se ela precisar ser trazida do disco. Terceiro, é possível que o programa apenas tenha acessado um endereço inválido e nenhum mapeamento precisa ser acrescentado à TLB. Nesse caso, o sistema operacional tipicamente mata o programa com uma falta de segmentação. Apenas nesse caso o programa fez algo errado. Todos os outros casos são automaticamente corrigidos pelo hardware e/ou o sistema operacional — ao custo de algum desempenho.

# Tabelas de páginas para memórias grandes

## Tabelas de páginas multinível

Evitar manter todas as tabelas de páginas na memória o tempo inteiro. Em particular, aquelas que não são necessárias não devem ser mantidas.

## Tabelas de páginas invertidas

Alternativa para os níveis cada vez maiores em uma hierarquia de paginação. Nesse projeto, há apenas uma entrada por quadro de página na memória real, em vez de uma entrada por página de espaço de endereço virtual. São comuns em máquinas de 64 bits porque mesmo com um tamanho de página muito grande, o número de entradas de tabela de páginas é gigantesco.

Nesse projeto, há apenas uma entrada por quadro de página na memória real, em vez de uma entrada por página de espaço de endereço virtual.

Embora tabelas de páginas invertidas poupem muito espaço, pelo menos quando o espaço de endereço virtual é muito maior do que a memória física, elas têm um sério problema: a tradução virtual-física torna-se muito mais difícil.

# Algoritmos de substituição de páginas

Quando ocorre uma falta de página, o sistema operacional tem de escolher uma página para remover da memória a fim de abrir espaço para a que está chegando. Se a página a ser removida foi modificada enquanto estava na memória, ela precisa ser reescrita para o disco a fim de atualizar a cópia em disco. Se, no entanto, ela não tiver sido modificada (por exemplo, ela contém uma página de código), a cópia em disco já está atualizada, portanto não é preciso reescrevê-la. A página a ser lida simplesmente sobrescreve a página que está sendo removida.

## Ótimo

Substitui a página que será referenciada mais tarde (no futuro). Impossível de implementar.

O algoritmo ótimo diz que a página com o maior rótulo deve ser removida. Se uma página não vai ser usada para 8 milhões de instruções e outra página não vai ser usada para 6 milhões de instruções, remover a primeira adia ao máximo a próxima falta de página. Computadores, como as pessoas, tentam adiar ao máximo a ocorrência de eventos desagradáveis.

No momento da falta de página, o sistema operacional não tem como saber quando cada uma das páginas será referenciada em seguida.

## Não usada recentemente (NRU – Not Recently Used)

Substitui a página que foi referenciada há mais tempo, considerando os bits R (referencia) e M (modificada).

Classe 0: não referenciada, não modificada;
Classe 1: não referenciada, modificada;
Classe 2: referenciada, não modificada;
Classe 3: referenciada, modificada;

Implícito nesse algoritmo está a ideia de que é melhor remover uma página modificada, mas não referenciada, a pelo menos um tique do relógio (em geral em torno de 20 ms) do que uma página não modificada que está sendo intensamente usada. A principal atração do NRU é que ele é fácil de compreender, moderadamente eficiente de implementar e proporciona um desempenho que, embora não ótimo, pode ser adequado.

Seleciona aleatoriamente a página da Classe mais baixa.

## Primeiro a Entrar, Primeiro a Sair (FIFO – First In, First Out)

Substitui a página mais antiga na memória principal. Quando aplicado aos computadores, surge o mesmo problema: a página mais antiga ainda pode ser útil. Por essa razão, o FIFO na sua forma mais pura raramente é usado.

## Segunda Chance

Modificação do FIFO que evita o problema de jogar fora uma página intensamente usada é inspecionar o bit R da página mais antiga. Utilizando o bit de referência (R). Se R = 0, substitui; se R = 1 a página vai para o final da fila (como se tivesse sido carregada naquele momento). O bit R é também colocado em 0.

O que o algoritmo segunda chance faz é procurar por uma página antiga que não esteja referenciada no intervalo de relógio mais recente. Se todas as páginas foram referenciadas, a segunda chance degenera-se em um FIFO puro.

## Do Relógio

Embora a segunda chance seja um algoritmo razoável, ele é desnecessariamente ineficiente, pois ele está sempre movendo páginas em torno de sua lista. Uma abordagem melhor é manter todos os quadros de páginas em uma lista circular na forma de um relógio. Um ponteiro aponta para a página mais antiga.

Modificação do Segunda Chance, onde é mantida uma lista circular, com um ponteiro para a página mais antiga. Quando ocorre uma falta de página, se o bit R = 0, a página apontada é removida. Se R = 1, R é setado para zero e o ponteiro passa a apontar para a próxima página na lista e repete o processo, até encontrar uma página com R = 0.

## Usada menos recentemente (LRU)

Uma boa aproximação para o algoritmo ótimo é baseada na observação de que as páginas que foram usadas intensamente nas últimas instruções provavelmente o serão em seguida de novo. De maneira contrária, páginas que não foram usadas há eras provavelmente seguirão sem ser utilizadas por um longo tempo. Essa ideia sugere um algoritmo realizável: quando ocorre uma falta de página, jogue fora aquela que não tem sido usada há mais tempo. Substitui a página referenciada há mais tempo.

Embora o LRU seja teoricamente realizável, ele não é nem um pouco barato. Para se implementar por completo o LRU, é necessário que seja mantida uma lista encadeada de todas as páginas na memória, com a página mais recentemente usada na frente e a menos recentemente usada na parte de trás. A dificuldade é que a lista precisa ser atualizada a cada referência de memória. Encontrar uma página na lista, deletá-la e então movê-la para

a frente é uma operação que demanda muito tempo, mesmo em hardware (presumindo que um hardware assim possa ser construído).

● Hw – Registrador 64 bits : Após cada referência de memória, o valor atual de Hw é armazenado na entrada da tabela de páginas para a página recém-referenciada. Quando ocorre uma falta de página, o sistema operacional examina todos os contadores na tabela de página para encontrar a mais baixa. Essa página é a usada menos recentemente.

## Simulação do LRU em software (NFU)

Implementação em software do LRU. Utiliza um contador para cada página e substitui páginas não usadas frequentemente.

A cada interrupção de relógio o bit R é adicionado a um contador (memória de elefante). Os contadores controlam mais ou menos quão frequentemente cada página foi referenciada. Quando ocorre uma falta de página, aquela com o contador mais baixo é escolhida para substituição.

O principal problema com o NFU é que ele lembra um elefante: jamais esquece nada. Por exemplo, em um compilador de múltiplos passos, as páginas que foram intensamente usadas durante o passo 1 podem ainda ter um contador alto bem adiante. Na realidade, se o passo 1 possuir o tempo de execução mais longo de todos os passos, as páginas contendo o código para os passos subsequentes poderão ter sempre contadores menores do que as páginas do passo 1. Em consequência, o sistema operacional removerá as páginas úteis em vez das que não estão mais sendo usadas.

Outra alternativa usando o bit R, considerando os últimos k períodos de tempo (algoritmo do envelhecimento).

## Conjunto de trabalho

Os processos apresentam uma localidade de referência, significando que durante qualquer fase de execução o processo referencia apenas uma fração relativamente pequena das suas páginas. Cada passo de um compilador de múltiplos passos, por exemplo, referencia apenas uma fração de todas as páginas, e a cada passo essa fração é diferente. O conjunto de páginas que um processo está atualmente usando é o seu conjunto de trabalho.

Se a memória disponível é pequena demais para conter todo o conjunto de trabalho, o processo causará muitas faltas de páginas e será executado lentamente, já que executar uma instrução leva alguns nanossegundos e ler em uma página a partir do disco costuma levar 10 ms. A um ritmo de uma ou duas instruções por 10 ms, seria necessária uma eternidade para terminar. Um programa causando faltas de páginas a todo o momento está ultrapaginando (thrashing).

Muitos sistemas de paginação tentam controlar o conjunto de trabalho de cada processo e certificar-se de que ele está na memória antes de deixar o processo ser executado. Essa abordagem é chamada de modelo do conjunto de trabalho. Ele foi projetado para reduzir substancialmente o índice de faltas de páginas. Carregar as páginas antes de deixar um processo ser executado também é chamado de pré- -paginação.

Há muito tempo se sabe que os programas raramente referenciam seu espaço de endereçamento de modo uniforme, mas que as referências tendem a agrupar-se em um pequeno número de páginas. Uma referência de memória pode buscar uma instrução ou dado, ou ela pode armazenar dados. Em qualquer instante de tempo, t, existe um conjunto consistindo de todas as páginas usadas pelas k referências de memória mais recentes. Esse conjunto, w(k, t), é o conjunto de trabalho. Como todas as k = 1 referências mais recentes precisam ter utilizado páginas que tenham sido usadas pelas k > 1 referências mais recentes, e possivelmente outras, w(k, t) é uma função monoliticamente não decrescente como função de k. À medida que k torna- -se grande, o limite de w(k, t) é finito, pois um programa não pode referenciar mais páginas do que o seu espaço de endereçamento contém, e poucos programas usarão todas as páginas. A Figura 3.18 descreve o tamanho do conjunto de trabalho como uma função de k.

A quantidade de tempo de CPU que um processo realmente usou desde que foi inicializado é muitas vezes chamada de seu tempo virtual atual.

A ideia básica é encontrar uma página que não esteja no conjunto de trabalho e removê-la. Como somente as páginas localizadas na memória são consideradas candidatas à remoção, as que estão ausentes da memória são ignoradas por esse algoritmo. Cada entrada contém (ao menos) dois itens fundamentais de informação: o tempo (aproximado) que a página foi usada pela última vez e o bit R (Referenciada).

O algoritmo funciona da seguinte maneira: supõe-se que o hardware inicializa os bits R e M, como já discutido. De modo similar, presume-se que uma interrupção periódica de relógio ative a execução do software que limpa o bit Referenciada em cada tique do relógio. A cada falta de página, a tabela de páginas é varrida à procura de uma página adequada para ser removida.

À medida que cada entrada é processada, o bit R é examinado. Se ele for 1, o tempo virtual atual é escrito no campo Instante de último uso na tabela de páginas, indicando que a página estava sendo usada no momento em que a falta ocorreu. Tendo em vista que a página foi referenciada durante a interrupção de relógio atual, ela claramente está no conjunto de trabalho e não é candidata a ser removida.

Se R é 0, a página não foi referenciada durante a interrupção de relógio atual e pode ser candidata à remoção. Para ver se ela deve ou não ser removida, sua idade (o tempo virtual atual menos seu Instante de último uso) é calculada e comparada a $\tau$. Se a idade for maior que $\tau$, a página não está mais no conjunto de trabalho e a página nova a substitui. A atualização das entradas restantes é continuada.

No entanto, se R é 0 mas a idade é menor do que ou igual a $\tau$, a página ainda está no conjunto de trabalho. A página é temporariamente poupada, mas a página com a maior

idade (menor valor de Instante do último uso) é marcada. Se a tabela inteira for varrida sem encontrar uma candidata para remover, isso significa que todas as páginas estão no conjunto de trabalho. Nesse caso, se uma ou mais páginas com R = 0 forem encontradas, a que tiver a maior idade será removida.

### WSClock

Implementa a ideias dos algoritmos do relógio e do conjunto de trabalho. Amplamente usado (simples e eficiente). Usa lista circular (relógio) e instante do último uso (conj. trab.)

Cada entrada contém o campo do Instante do último uso do algoritmo do conjunto de trabalho básico, assim como o bit R (mostrado) e o bit M (não mostrado).

Uso dos bits R, M, do instante do último uso e $\tau$ ("tamanho" do tamanho" do conjunto de trabalho)

Se R = 1, seta R = 0 e avança para a próxima página;

Se R = 0 e tempo > $\tau$:
Não modificada → usa o quadro (substitui a página);
Modificada – continua a busca (mas pode ser a vítima);

Se R = 0 e tempo < $\tau$;
Continua a busca;

O que acontece se o ponteiro deu uma volta completa e voltou ao seu ponto de partida? Há dois casos que precisamos considerar:

1. Pelo menos uma escrita foi escalonada.
2. Nenhuma escrita foi escalonada.

No primeiro caso, o ponteiro apenas continua a se mover, procurando por uma página limpa. Dado que uma ou mais escritas foram escalonadas, eventualmente alguma escrita será completada e a sua página marcada como limpa. A primeira página limpa encontrada é removida.

No segundo caso, todas as páginas estão no conjunto de trabalho, de outra maneira pelo menos uma escrita teria sido escalonada. Por falta de informações adicionais, a coisa mais simples a fazer é reivindicar qualquer página limpa e usá-la. A localização de uma página limpa pode ser registrada durante a varredura. Se não existir nenhuma, então a página atual é escolhida como a vítima e será reescrita em disco.

## Resumo dos Algoritmos de substituição de páginas

Ótimo: Não implementável, mas útil como um padrão de desempenho;
NRU (não usado recentemente): Aproximação muito rudimentar do LRU
FIFO (primeiro a entrar, primeiro a sair): Pode descartar páginas importantes
Segunda chance: Algoritmo FIFO bastante melhorado
Relógio: Realista
LRU (usada menos recentemente): Excelente algoritmo, porém difícil de ser implementado de maneira exata
NFU (não frequentemente usado): Aproximação bastante rudimentar do LRU
Envelhecimento (aging): Algoritmo eficiente que aproxima bem o LRU
Conjunto de trabalho: Implementação um tanto cara
WSClock: Algoritmo bom e eficiente

## Políticas de alocação local versus global

Algoritmos locais correspondem a alocar a todo processo uma fração fixa da memória. Algoritmos globais alocam dinamicamente quadros de páginas entre os processos executáveis. Desse modo, o número de quadros de páginas designadas a cada processo varia com o tempo.

Em geral, algoritmos globais funcionam melhor, especialmente quando o tamanho do conjunto de trabalho puder variar muito através do tempo de vida de um processo.

Outra abordagem é ter um algoritmo para alocar quadros de páginas para processos. Pode-se determinar periodicamente o número de processos em execução e alocar a cada processo uma porção igual. Desse modo, com 12.416 quadros de páginas disponíveis (isto é, sistema não operacional) e 10 processos, cada processo recebe 1.241 quadros. Os seis restantes vão para uma área comum a ser usada quando ocorrer a falta de página.

Uma maneira de gerenciar a alocação é usar o algoritmo PFF (Page Fault Frequency — frequência de faltas de página). Ele diz quando aumentar ou diminuir a alocação de páginas de um processo, mas não diz nada sobre qual página substituir em uma falta. Ele apenas controla o tamanho do conjunto de alocação.

Medir a frequência de faltas de página é algo direto: apenas conte o número de faltas por segundo, possivelmente tomando a média de execução através dos últimos segundos também.

A escolha de local versus global, em alguns casos, é independente do algoritmo. Por outro lado, para outros algoritmos de substituição de página, apenas uma estratégia local faz sentido. Em particular, o conjunto de trabalho e os algoritmos WSClock referem-se a algum processo específico e devem ser aplicados nesse contexto. Na realidade, não há um conjunto de trabalho para a máquina como um todo, e tentar usar a união de todos os conjuntos de trabalho perderia a propriedade de localidade e não funcionaria bem.

## Controle de carga

Caso ocorra uma ultrapaginação, será necessário dar mais memória a alguns processos. Pelo fato de não haver um modo de dar mais memória àqueles processos que precisam dela sem prejudicar alguns outros, a única solução real é livrar-se temporariamente de alguns processos. Na realidade, sempre que os conjuntos de trabalho combinados de todos os processos excedem a capacidade da memória, a ultrapaginação pode ser esperada.

Uma boa maneira de reduzir o número de processos competindo pela memória é levar alguns deles para o disco e liberar todas as páginas que eles estão segurando.

É preciso considerar também o grau de multiprogramação, ou seja, o tamanho dos processos e frequência da paginação ao decidir qual processo deve ser trocado, bem como as características.

## Tamanho de página

É um parâmetro que pode ser escolhido pelo sistema operacional. Determinar o melhor tamanho de página exige equilibrar vários fatores, pois não há um tamanho ótimo geral.

Para começo de conversa, dois fatores pedem um tamanho de página pequeno. Um segmento de código, dados, ou pilha escolhido ao acaso não ocupará um número inteiro de páginas. Na média, metade da página final estará vazia. O espaço extra nessa página é desperdiçado. O espaço extra de uma página que é desperdiçado é denominado fragmentação interna.

Outro argumento em defesa de um tamanho de página pequeno torna-se aparente quando pensamos sobre um programa consistindo em oito fases sequenciais de 4 KB cada. Com um tamanho de página de 32 KB, esse programa demandará 32 KB durante o tempo inteiro de execução. Com um tamanho de página de 16 KB, ele precisará de apenas 16 KB. Com um tamanho de página de 4 KB ou menor, ele exigirá apenas 4 KB a qualquer instante. Em geral, um tamanho de página grande causará mais desperdício de espaço na memória.

Um tamanho de página grande pode causar mais desperdício de espaço na memória. Páginas pequenas ocupam muito espaço no TLB. Para equilibrar essas escolhas, sistemas operacionais às vezes usam tamanhos diferentes de páginas para partes diferentes do sistema. Por exemplo, páginas grandes para o núcleo e menores para os processos do usuário.

## Espaços separados de instruções e dados

Se o espaço de endereçamento for grande o suficiente, tudo funcionará bem. Se for pequeno demais, ele força os programadores a encontrar uma saída para fazer caber tudo nesse espaço.

Uma solução é ter dois espaços de endereçamento diferentes para instruções (código do programa) e dados, chamados de espaço I e espaço D.

O ligador (linker) precisa saber quando endereços I e D separados estão sendo usados, pois quando eles estão, os dados são realocados para o endereço virtual 0, em vez de começarem após o programa.

Em um computador com esse tipo de projeto, ambos os espaços de endereçamento podem ser paginados, independentemente um do outro. Cada um tem sua própria tabela de páginas, com seu próprio mapeamento de páginas virtuais para quadros de página física. Quando o hardware quer buscar uma instrução, ele sabe que deve usar o espaço I e a tabela de páginas do espaço I. De modo similar, dados precisam passar pela tabela de páginas do espaço D. Fora essa distinção, ter espaços de I e D separados não apresenta quaisquer complicações especiais para o sistema operacional e duplica o espaço de endereçamento disponível.

Embora os espaços de endereçamento sejam grandes, seu tamanho costumava ser um problema sério. Mesmo hoje, no entanto, espaços de I e D separados são comuns. No entanto, em vez de serem usados para os espaços de endereçamento normais, eles são usados agora para dividir a cache L1. Afinal de contas, na cache L1, a memória ainda é bastante escassa.

## Páginas compartilhadas

Outra questão de projeto importante é o compartilhamento. Em um grande sistema de multiprogramação, é comum que vários usuários estejam executando o mesmo programa ao mesmo tempo.

É eficiente compartilhar as páginas para evitar ter duas cópias da mesma página na memória ao mesmo tempo. Pesquisar todas as tabelas de páginas para ver se uma página está sendo compartilhada normalmente é muito caro, portanto estruturas de dados especiais são necessárias para controlar as páginas compartilhadas, em especial se a unidade de compartilhamento for a página individual (ou conjunto de páginas), em vez de uma tabela de páginas inteira.

Se o sistema der suporte aos espaços I e D, o compartilhamento de programas é algo relativamente direto, fazendo que dois ou mais processos usem a mesma tabela de páginas para seu espaço I, mas diferentes tabelas de páginas para seus espaços D.

Quando dois ou mais processos compartilham algum código, um problema ocorre com as páginas compartilhadas. Suponha que os processos A e B estejam ambos executando o editor e compartilhando suas páginas. Se o escalonador decidir remover A da memória, removendo todas as suas páginas e preenchendo os quadros das páginas vazias com

algum outro programa, isso fará que B gere um grande número de faltas de páginas para trazê-las de volta outra vez.

## Bibliotecas compartilhadas

São chamadas de DLLs ou Dynamic Link Libraries — Bibliotecas de Ligação Dinâmica — no Windows).

Além de tornar arquivos executáveis menores e também salvar espaço na memória, têm outra vantagem importante: se uma função em uma biblioteca compartilhada for atualizada para remover um erro, não será necessário recompilar os programas que a chamam e os antigos arquivos binários continuam a funcionar.

Desvantagem: a realocação durante a execução não funciona. É preciso usar o método copiar na escrita e criar novas páginas para cada processo compartilhando a biblioteca, realocando-as dinamicamente quando são criadas.

Ex: No processo 1, a biblioteca começa no endereço 36K; no processo 2, em 12K. Suponha que a primeira coisa que a primeira função na biblioteca tem de fazer é saltar para o endereço 16 na biblioteca. No entanto, como a biblioteca é compartilhada, a realocação durante a execução não funcionará. Afinal, quando a primeira função é chamada pelo processo 2 (no endereço 12K), a instrução de salto tem de ir para 12K + 16, não 36K + 16. Esse é o pequeno problema. Uma maneira de solucioná-lo é usar o método copiar na escrita e criar novas páginas para cada processo compartilhando a biblioteca, realocando-as dinamicamente quando são criadas, mas esse esquema obviamente frustra o propósito de compartilhamento da biblioteca.

Uma solução melhor é compilar bibliotecas compartilhadas com uma flag de compilador especial dizendo ao compilador para não produzir quaisquer instruções que usem endereços absolutos. Em vez disso, apenas instruções usando endereços relativos são usadas. Por exemplo, quase sempre há uma instrução que diz “salte para a frente” (ou para trás) n bytes (em oposição a uma instrução que dá um endereço específico para saltar). Essa instrução funciona corretamente não importa onde a biblioteca compartilhada estiver colocada no espaço de endereço virtual. Ao evitar endereços absolutos, o problema pode ser solucionado. O código que usa apenas deslocamentos relativos é chamado de código independente do posicionamento.

## Arquivos mapeados

Trata-se de um processo que pode emitir uma chamada de sistema para mapear um arquivo em uma porção do seu espaço virtual.

Em vez de fazer leituras e gravações, o arquivo pode ser acessado como um grande arranjo de caracteres na memória. Em algumas situações, os programadores consideram esse modelo mais conveniente.

Se dois ou mais processos mapeiam o mesmo arquivo ao mesmo tempo, eles podem comunicar-se via memória compartilhada. Gravações feitas por um processo a uma memória compartilhada são imediatamente visíveis quando o outro lê da parte de seu espaço de endereçamento virtual mapeado no arquivo. Desse modo, esse mecanismo fornece um canal de largura de banda elevada entre processos e é muitas vezes usado como tal (a ponto de mapear até mesmo um arquivo temporário).

As bibliotecas podem usar esse mecanismo se arquivos mapeados em memória estiverem disponíveis.

# Política de limpeza

Os sistemas de paginação geralmente têm um processo de segundo plano, chamado de daemon de paginação que, periodicamente, inspeciona o estado da memória e garante que todos os quadros estejam limpos, assim eles não precisam ser escritos às pressas para o disco quando requisitados.

Uma maneira de implementar essa política de limpeza é com um relógio de dois ponteiros. O ponteiro da frente é controlado pelo daemon de paginação. Quando ele aponta para uma página suja, ela é reescrita para o disco e o ponteiro da frente é avançado. Quando aponta para uma página limpa, ele simplesmente avança. O ponteiro de trás é usado para a substituição de página, como no algoritmo de relógio-padrão. Apenas agora, a probabilidade do ponteiro de trás apontar para uma página limpa é aumentada em virtude do trabalho do daemon de paginação.

# Questões de implementação

## Envolvimento do sistema operacional com a paginação

Há quatro momentos em que o sistema operacional tem de se envolver com a paginação: na criação do processo, na execução do processo, em faltas de páginas e no término do processo.

Quando um novo processo é criado em um sistema de paginação, o sistema operacional precisa determinar qual o tamanho que o programa e os dados terão (de início) e criar uma tabela de páginas para eles.

Quando um processo é escalonado para execução, a MMU tem de ser reinicializada para o novo processo e o TLB descarregado para se livrar de traços do processo que estava sendo executado. A tabela de páginas do novo processo deve tornar-se a atual, normalmente copiando a tabela ou um ponteiro para ela em algum(ns) registrador(es) de hardware.

Quando ocorre uma falta de página, o sistema operacional precisa ler registradores de hardware para determinar quais endereços virtuais a causaram. A partir dessa informação, ele deve calcular qual página é necessária e localizá-la no disco. Ele deve então encontrar um quadro de página disponível no qual colocar a página nova, removendo alguma página antiga se necessário. Depois ele precisa ler a página necessária para o quadro de página. Por fim, tem de salvar o contador do programa para que ele aponte para a instrução que causou a falta de página e deixar que a instrução seja executada novamente.

Quando um processo termina, o sistema operacional deve liberar a sua tabela de páginas, suas páginas e o espaço de disco que elas ocupam quando estão no disco. Se algumas das páginas forem compartilhadas com outros processos, as páginas na memória e no disco poderão ser liberadas somente quando o último processo que as estiver usando for terminado.

## Backup de instrução

Backup de instrução: quando um programa referencia uma página que não está na memória, a instrução que causou a falta é parada no meio da sua execução e ocorre uma interrupção para o sistema operacional. Após o sistema operacional ter buscado a página necessária, ele deve reiniciar a instrução que causou a interrupção.

## Retenção de páginas na memória

Se um dispositivo de E/S estiver no processo de realizar uma transferência via DMA para aquela página, removê-lo fará que parte dos dados seja escrita no buffer a que eles pertencem, e parte seja escrita sobre a página recém-carregada. Uma solução para esse problema é trancar as páginas engajadas em E/S na memória de maneira que elas não sejam removidas. Trancar uma página é muitas vezes chamado de fixação (pinning) na memória.

## Gerenciamento de disco

O algoritmo mais simples para alocação de espaço em disco consiste na criação de uma área de troca especial no disco ou, até melhor, em um disco separado do sistema de arquivos (para equilibrar a carga de E/S).

No entanto, esse modelo simples tem um problema: processos podem aumentar em tamanho após serem iniciados. Embora o programa de texto seja normalmente fixo, a área de dados pode às vezes crescer, e a pilha pode sempre crescer. Em consequência, pode ser melhor reservar áreas de troca separadas para o texto, dados e pilha e permitir que cada uma dessas áreas consista de mais do que um pedaço do disco.

O outro extremo é não alocar nada antecipadamente e alocar espaço de disco para cada página quando ela for removida e liberar o mesmo espaço quando ela for carregada na memória. Dessa maneira, os processos na memória não ficam amarrados a qualquer

espaço de troca. A desvantagem é que um endereço de disco é necessário na memória para controlar cada página no disco. Em outras palavras, é preciso haver uma tabela por processo dizendo para cada página no disco onde ela está

### Separação da política e do mecanismo

Esse princípio pode ser aplicado ao gerenciamento da memória fazendo que a maioria dos gerenciadores de memória seja executada como processos no nível do usuário.

## Segmentação

Fornecer à máquina espaços de endereçamento completamente independentes, que são chamados de segmentos. Ajuda a lidar com estruturas de dados que podem mudar de tamanho durante a execução e simplifica a ligação e o compartilhamento. Cada segmento consiste em uma sequência linear de endereços, começando em 0 e indo até algum valor máximo. O comprimento de cada segmento pode ser qualquer coisa de 0 ao endereço máximo permitido. Diferentes segmentos podem e costumam ter comprimentos diferentes. Além disso, comprimentos de segmentos podem mudar durante a execução. O comprimento de um segmento de pilha pode ser aumentado sempre que algo é colocado sobre a pilha e diminuído toda vez que algo é retirado dela.

Claro, um segmento pode ficar cheio, mas segmentos em geral são muito grandes, então essa ocorrência é rara. Para especificar um endereço nessa memória segmentada ou bidimensional, o programa precisa fornecer um endereço em duas partes, um número de segmento e um endereço dentro do segmento.

Enfatizamos aqui que um segmento é uma entidade lógica, que o programador conhece e usa como uma entidade lógica. Um segmento pode conter uma rotina, um arranjo, uma pilha, ou um conjunto de variáveis escalares, mas em geral ele não contém uma mistura de tipos diferentes.

Facilita proporcionar proteção para diferentes segmentos. Às vezes a segmentação e a paginação são combinadas para proporcionar uma memória virtual bidimensional.

## Segmentação X Paginação

O programador precisa saber que essa técnica está sendo usada?
Paginação: Não
Segmentação: Sim

Há quantos espaços de endereçamento linear?
Paginação: 1
Segmentação: Muitos

O espaço de endereçamento total pode superar o tamanho da memória física?
Paginação: Sim
Segmentação: Sim

Rotinas e dados podem ser distinguidos e protegidos separadamente?
Paginação: Não
Segmentação: Sim

As tabelas cujo tamanho flutua podem ser facilmente acomodadas?
Paginação:  Não
Segmentação: Sim

O compartilhamento de rotinas entre os usuários é facilitado?
Paginação: Não
Segmentação: Sim

Por que essa técnica foi inventada?
Paginação: Para obter um grande espaço de endereçamento linear sem a necessidade de comprar mais memória física
Segmentação:  Para permitir que programas e dados sejam divididos em espaços de endereçamento logicamente independentes e para auxiliar o compartilhamento e a proteção

## Segmentação com paginação

Se os segmentos forem grandes, talvez não seja possível mantê-los na memória principal em sua totalidade. Isso leva à ideia de realizar a paginação dos segmentos, de maneira que apenas aquelas páginas de um segmento que são realmente necessárias tenham de estar na memória.

O sistema MULTICS e o x86 de 32 bits da Intel dão suporte à segmentação e à paginação. Ainda assim, fica claro que poucos projetistas de sistemas operacionais se preocupam mesmo com a segmentação (pois são casados a um modelo de memória diferente). Em consequência, ela parece estar saindo rápido de moda. Hoje, mesmo a versão de 64 bits do x86 não dá mais suporte a uma segmentação de verdade.

Para implementá-lo, os projetistas do MULTICS escolheram tratar cada segmento como uma memória virtual e assim paginá-lo, combinando as vantagens da paginação (tamanho da página uniforme e não precisar manter o segmento todo na memória caso apenas uma parte dele estivesse sendo usada) com as vantagens da segmentação (facilidade de programação, modularidade, proteção, compartilhamento).

Um descritor de segmento continha um indicativo sobre se o segmento estava na memória principal ou não. Se qualquer parte do segmento estivesse na memória, ele era considerado estando na memória, e sua tabela de página estaria.

Um endereço no MULTICS consistia em duas partes: o segmento e o endereço dentro dele. O endereço dentro do segmento era dividido ainda em um número de página e uma palavra dentro da página.

• Espaço de endereçamento virtual
    - 256 K segmentos de 64 K palavras de 36 bits
    - Endereço virtual de 34 bits
    - Páginas de 1K palavras

• Máquina com endereço físico de 24 bits

Como você deve ter percebido, se o algoritmo anterior fosse de fato utilizado pelo sistema operacional em todas as instruções, os programas não executariam rápido. Na realidade, o hardware MULTICS continha um TLB de alta velocidade de 16 palavras que podia pesquisar todas as suas entradas em paralelo para uma dada chave. Esse foi o primeiro sistema a ter um TLB algo usado em todas as arquiteturas modernas.

Os endereços das 16 páginas mais recentemente referenciadas eram mantidos no TLB. Programas cujo conjunto de trabalho era menor do que o tamanho do TLB encontravam equilíbrio com os endereços de todo o conjunto de trabalho no TLB e, portanto, executavam eficientemente; de outra forma, ocorriam faltas no TLB.

## Segmentação com paginação no Intel x86 - 32 bits

• Espaço de endereçamento virtual
    - 16 K segmentos de 1 M palavras de 32 bits
    - Endereço virtual de 32 bits
    - Páginas de 4K palavras

• Máquina com endereço físico de 32 bits

Embora existam menos segmentos, o tamanho maior deles é muito mais importante, à medida que poucos programas precisam de mais de 1000 segmentos, mas muitos programas precisam de grandes segmentos.

O coração da memória virtual do x86 consiste em duas tabelas, chamadas de LDT (Local Descriptor Table — tabela de descritores locais) e GDT (Global Descriptor Table — tabela de descritores globais). Cada programa tem seu próprio LDT, mas há uma única GDT, compartilhada por todos os programas no computador. A LDT descreve segmentos locais a cada programa, incluindo o seu código, dados, pilha e assim por diante, enquanto a GDT descreve segmentos de sistema, incluindo o próprio sistema operacional.

Para acessar um segmento, um programa x86 primeiro carrega um seletor para aquele segmento em um dos seis registradores de segmentos da máquina.

No momento em que um seletor é carregado em um registrador de segmento, o descritor correspondente é buscado na LDT ou na GDT e armazenado em registradores de microprogramas, de maneira que eles possam ser acessados rapidamente.

Presumindo que o segmento está na memória e o deslocamento está dentro do alcance, o x86 então acrescenta o campo Base de 32 bits no descritor ao deslocamento para formar o que é chamado de endereço linear.

Se a paginação for desabilitada (por um bit em um registrador de controle global), o endereço linear será interpretado como o endereço físico e enviado para a memória para ser lido ou escrito. Desse modo, com a paginação desabilitada, temos um esquema de segmentação puro, com cada endereço base do segmento dado em seu descritor. Segmentos não são impedidos de sobrepor-se, provavelmente porque daria trabalho demais verificar se eles estão disjuntos.

Por outro lado, se a paginação estiver habilitada, o endereço linear é interpretado como um endereço virtual e mapeado no endereço físico usando as tabelas de páginas, de maneira bastante semelhante aos nossos exemplos anteriores. A única complicação real é que com um endereço virtual de 32 bits e uma página de 4 KB, um segmento poderá conter 1 milhão de páginas, então um mapeamento em dois níveis é usado para reduzir o tamanho da tabela de páginas para segmentos pequenos.

Cada programa em execução tem um diretório de páginas consistindo de 1024 entradas de 32 bits. Ele está localizado em um endereço apontado por um registrador global. Cada entrada nesse diretório aponta para uma tabela de páginas também contendo 1024 entradas de 32 bits. As entradas da tabela de páginas apontam para os quadros de páginas.

Para evitar fazer repetidas referências à memória, o x86, assim como o MULTICS, tem uma TLB pequena que mapeia diretamente as combinações Dir-Página mais recentemente usadas no endereço físico do quadro de página. Apenas quando a combinação atual não estiver presente na TLB o mecanismo será realmente executado e a TLB atualizada. Enquanto as faltas na TLB forem raras, o desempenho será bom.

Vale a pena observar que se alguma aplicação não precisa de segmentação, mas está simplesmente contente com um único espaço de endereçamento paginado de 32 bits, o modelo citado é possível. Todos os registradores de segmentos podem ser estabelecidos com o mesmo seletor, cujo descritor tem Base = 0 e Limite estabelecido para o máximo. O deslocamento da instrução será então o endereço linear, com apenas um único espaço de endereçamento usado — na realidade, paginação normal.

## Pesquisa em gerenciamento de memória

Hoje, a pesquisa sobre a paginação se concentra em tipos novos de sistemas.

Paginação em sistemas com múltiplos núcleos: esses tipos de sistemas tendem a possuir muita memória cache compartilhada de maneiras complexas.

Paginação em sistemas NUMA: diversas partes da memória podem ter diferentes tempos de acesso.

Smartphones e tablets.

Sistemas em tempo real.