As caches de segundo nível se tornaram comuns quando os projetistas descobriram que o silício limitado e as metas de altas velocidades de clock impedem que as caches primárias se tornem grandes. A cache secundária, que normalmente é 10 ou mais vezes maior do que a cache primária, trata muitos acessos que falham na cache primária. Nesses casos, a penalidade de falha é aquela do tempo de acesso à cache secundária (em geral, menos de dez ciclos de processador) contra o tempo de acesso à memória (normalmente mais de 100 ciclos de processador). Assim como na associatividade, as negociações de projeto entre o tamanho da cache secundária e seu tempo de acesso dependem de vários aspectos de implementação.

…foi inventado um sistema para fazer a combinação entre os sistemas centrais de memória e os tambores de discos aparecer para o programador como um depósito de nível único, com as transferências necessárias ocorrendo automaticamente.

Kilburn et al., *One-level storage system, 1962*

5.4 Memória Virtual

Na seção anterior, vimos como as caches fornecem acesso rápido às partes recentemente usadas do código e dos dados de um programa. Da mesma forma, a memória principal pode agir como uma "cache" para o armazenamento secundário, normalmente implementado com discos magnéticos. Essa técnica é chamada de **memória virtual**. Historicamente, houve duas motivações principais para a memória virtual: permitir o compartilhamento seguro e eficiente da memória entre vários programas e remover os transtornos de programação de uma quantidade pequena e limitada de memória principal. Quatro décadas após sua invenção, o primeiro motivo é o que ainda predomina.

memória virtual Uma técnica que usa a memória principal como uma "cache" para armazenamento secundário.

Considere um grupo de programas executados ao mesmo tempo em um computador. É claro que, para permitir que vários programas compartilhem a mesma memória, precisamos ser capazes de proteger os programas uns dos outros, garantindo que um programa só possa ler e escrever as partes da memória principal atribuídas a ele. A memória principal precisa conter apenas as partes ativas dos muitos programas, exatamente como uma cache contém apenas a parte ativa de um programa. Portanto, o princípio da localidade possibilita a memória virtual e as caches, e a memória virtual nos permite compartilhar eficientemente o processador e a memória principal.

Não podemos saber quais programas irão compartilhar a memória com outros programas quando os compilamos. Na verdade, os programas que compartilham a memória mudam dinamicamente enquanto estão sendo executados. Devido a essa interação dinâmica, gostaríamos de compilar cada programa para o seu próprio *espaço de endereçamento* – faixa distinta dos locais de memória acessível apenas a esse programa. A memória virtual implementa a tradução do espaço de endereçamento de um programa para os **endereços físicos**. Esse processo de tradução impõe a **proteção** do espaço de endereçamento de um programa contra outros programas.

endereço físico Um endereço na memória principal.

proteção Um conjunto de mecanismos para garantir que múltiplos processos compartilhando processador, memória ou dispositivos de E/S não possam interferir, intencionalmente ou não, um com o outro, lendo ou escrevendo dados no outro. Esses mecanismos também isolam o sistema operacional de um processo de usuário.

A segunda motivação para a memória virtual é permitir que um único programa do usuário exceda o tamanho da memória principal. Antigamente, se um programa se tornasse muito grande para a memória, cabia ao programador fazê-lo se adequar. Os programadores dividiam os programas em partes e, então, identificavam aquelas mutuamente exclusivas. Esses *overlays* eram carregados ou descarregados sob o controle do programa do usuário durante a execução, com o programador garantindo que o programa nunca tentaria acessar um overlay que não estivesse carregado e que os overlays carregados nunca excederiam o tamanho total da memória. Os overlays eram tradicionalmente organizados como módulos, cada um contendo código e dados. As chamadas entre procedimentos em módulos diferentes levavam um módulo a se sobrepor a outro.

Como você pode bem imaginar, essa responsabilidade era uma carga substancial para os programadores. A memória virtual, criada para aliviar os programas dessa dificuldade, gerencia automaticamente os dois níveis da hierarquia de memória representados pela memória principal (às vezes, chamada de *memória física* para distingui-la da memória virtual) e pelo armazenamento secundário.

Embora os conceitos aplicados na memória virtual e nas caches sejam os mesmos, suas diferentes raízes históricas levaram ao uso de uma terminologia diferente. Um bloco de memória virtual é chamado de *página,* e uma falha da memória virtual é chamada de **falta de página**. Com a memória virtual, o processador produz um **endereço virtual**, traduzido por uma combinação de hardware e software para um *endereço físico*, que, por sua vez, pode ser usado de modo a acessar a memória principal. A Figura 5.19 mostra a memória endereçada virtualmente com páginas mapeadas na memória principal. Esse processo é chamado de *mapeamento de endereço* ou **tradução de endereço**. Hoje, os dois níveis de hierarquia de memória controlados pela memória virtual são as DRAMs e os discos magnéticos (veja o Capítulo 1, Seção "Um lugar seguro para os dados"). Se voltarmos à nossa analogia da biblioteca, podemos pensar no endereço virtual como o título de um livro e no endereço físico como seu local na biblioteca.

falta de página Um evento que ocorre quando uma página acessada não está presente na memória principal.

endereço virtual Um endereço que corresponde a um local no espaço virtual e é traduzido pelo mapeamento de endereço para um endereço físico quando a memória é acessada.

tradução de endereço Também chamada de mapeamento de endereço. O processo pelo qual um endereço virtual é mapeado a um endereço usado para acessar a memória.

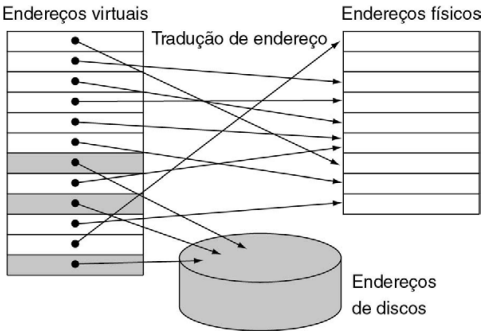

FIGURA 5.19 Na memória virtual, os blocos de memória (chamados de *páginas*) são mapeados de um conjunto de endereços (chamados de *endereços virtuais*) em outro conjunto (chamado de *endereços físicos*). O processador gera endereços virtuais enquanto a memória é acessada usando endereços físicos. Tanto a memória virtual quanto a memória física são desmembradas em páginas, de modo que uma página virtual é realmente mapeada em uma página física. Naturalmente, também é possível que uma página virtual esteja ausente da memória principal e não seja mapeada para um endereço físico, residindo no disco em vez disso. As páginas físicas podem ser compartilhadas fazendo dois endereços virtuais apontarem para o mesmo endereço físico. Essa capacidade é usada para permitir que dois programas diferentes compartilhem dados ou código.

A memória virtual também simplifica o carregamento do programa para execução fornecendo *relocação*. A relocação mapeia os endereços virtuais usados por um programa para diferentes endereços físicos antes que os endereços sejam usados no acesso à memória. Essa relocação nos permite carregar o programa em qualquer lugar na memória principal. Além disso, todos os sistemas de memória virtual em uso atualmente relocam o programa como um conjunto de blocos (páginas) de tamanho fixo, eliminando, assim, a necessidade de encontrar um bloco contíguo de memória para alocar um programa; em vez disso, o sistema operacional só precisa encontrar um número suficiente de páginas na memória principal.

Na memória virtual, o endereço é desmembrado em um *número de página virtual* e um *offset de página*. A Figura 5.20 mostra a tradução do número de página virtual para um *número de página física*. O número de página física constitui a parte mais significativa do endereço físico, enquanto o offset de página, que não é alterado, constitui a parte menos significativa. O número de bits no campo offset de página determina o tamanho da página. O número de páginas endereçáveis com o endereço virtual não precisa corresponder ao número de páginas endereçáveis com o endereço físico. Ter um número de páginas virtuais maior do que as páginas físicas é a base para a ilusão de uma quantidade de memória virtual essencialmente ilimitada.

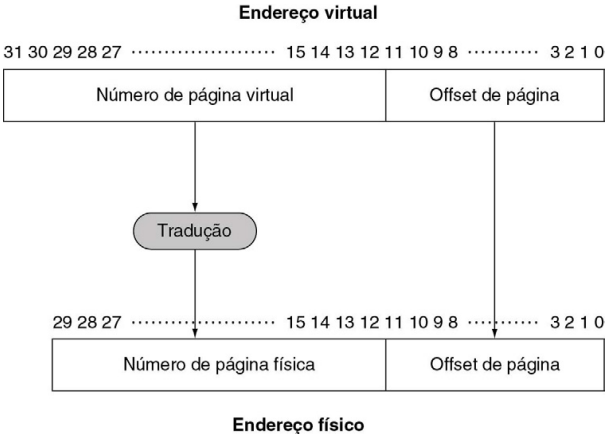

FIGURA 5.20 Mapeamento de um endereço virtual em um endereço físico. O tamanho de página é 2^{12} = 4KB. O número de páginas físicas permitido na memória é 2^{18}, já que o número de página física contém 18 bits. Portanto, a memória principal pode ter, no máximo, 1GB, enquanto o espaço de endereço virtual possui 4GB.

Muitas escolhas de projeto nos sistemas de memória virtual são motivadas pelo alto custo de uma falha, que, na memória virtual, tradicionalmente é chamada de falta de página. Uma falta de página levará milhões de ciclos de clock para ser processada. (A tabela na Seção 5.1 mostra que a memória principal é aproximadamente 100.000 vezes mais rápida do que o disco.) Essa enorme penalidade de falha, dominada pelo tempo para obter a primeira palavra para tamanhos de página típicos, leva a várias decisões importantes nos sistemas de memória virtual:

- As páginas devem ser grandes o suficiente para tentar amortizar o longo tempo de acesso. Tamanhos de 4KB a 16KB são comuns atualmente. Novos sistemas de desktop e servidor estão sendo desenvolvidos para suportar páginas de 32KB e 64KB, embora novos sistemas embutidos estejam indo na outra direção, para páginas de 1KB.
- Organizações que reduzem a taxa de faltas de página são atraentes. A principal técnica usada aqui é permitir o posicionamento totalmente associativo das páginas na memória.
- As faltas de página podem ser tratadas em nível de software porque o overhead será pequeno se comparado com o tempo de acesso ao disco. Além disso, o software pode se dar ao luxo de usar algoritmos inteligentes para escolher como posicionar as páginas, já que mesmo pequenas reduções na taxa de falhas compensarão o custo desses algoritmos.
- O write-through não funcionará para a memória virtual, visto que as escritas levam muito tempo. Em vez disso, os sistemas de memória virtual usam write-back.

As próximas subseções tratam desses fatores no projeto de memória virtual.

Detalhamento: Embora normalmente imaginemos os endereços virtuais como muito maiores do que os endereços físicos, o contrário pode ocorrer quando o tamanho de endereço do processador é pequeno em relação ao estado da tecnologia de memória. Nenhum programa único pode se beneficiar, mas um grupo de programas executados ao mesmo tempo pode se beneficiar de não precisar ser trocado para a memória ou de ser executado em processadores paralelos. Para computadores servidores e desktops, processadores de 32 bits já são problemáticos.

Detalhamento: A discussão da memória virtual neste livro focaliza a paginação, que usa blocos de tamanho fixo. Há também um esquema de blocos de tamanho variável chamado **segmentação**. Na segmentação, um endereço consiste em duas partes: um número de segmento e um offset de segmento. O registrador de segmento é mapeado a um endereço físico e o offset é *somado* para encontrar o endereço físico real. Como o segmento pode variar em tamanho, uma verificação de limites é necessária para garantir que o offset esteja dentro do segmento. O principal uso da segmentação é suportar métodos de proteção mais avançados e compartilhar um espaço de endereçamento. A maioria dos livros de sistemas operacionais contém extensas discussões sobre a segmentação comparada com a paginação e sobre o uso da segmentação para compartilhar logicamente o espaço de endereçamento. A principal desvantagem da segmentação é que ela divide o espaço de endereço em partes logicamente separadas que precisam ser manipuladas como um endereço de duas partes: o número de segmento e o offset. A paginação, por outro lado, torna o limite entre o número de página e o offset invisível aos programadores e compiladores.

segmentação Um esquema de mapeamento de endereço de tamanho variável em que um endereço consiste em duas partes: um número de segmento, que é mapeado para um endereço físico, e um offset de segmento.

Os segmentos também têm sido usados como um método para estender o espaço de endereçamento sem mudar o tamanho da palavra do computador. Essas tentativas têm sido malsucedidas devido à dificuldade e ao ônus de desempenho inerentes a um endereço de duas partes, dos quais os programadores e compiladores precisam estar cientes.

Muitas arquiteturas dividem o espaço de endereçamento em grandes blocos de tamanho fixo que simplificam a proteção entre o sistema operacional e os programas de usuário e aumentam a eficiência da paginação. Embora essas divisões normalmente sejam chamadas de "segmentos", esse mecanismo é muito mais simples do que a segmentação de tamanho de bloco variável e não é visível aos programas do usuário; discutiremos o assunto em mais detalhes em breve.

Posicionando uma página e a encontrando novamente

Em razão da penalidade incrivelmente alta decorrente de uma falta de página, os projetistas reduzem a frequência das faltas de página otimizando seu posicionamento. Se permitirmos que uma página virtual seja mapeada em qualquer página física, o sistema operacional, então, pode escolher substituir qualquer página que desejar quando ocorrer uma falta de página. Por exemplo, o sistema operacional pode usar um sofisticado algoritmo e complexas estruturas de dados, que monitoram o uso de páginas, para tentar escolher uma página que não será necessária por um longo tempo. A capacidade de usar um esquema de substituição inteligente e flexível reduz a taxa de faltas de página e simplifica o uso do posicionamento de páginas totalmente associativo.

Como mencionamos na Seção 5.3, a dificuldade em usar posicionamento totalmente associativo está em localizar uma entrada, já que ela pode estar em qualquer lugar no nível superior da hierarquia. Uma pesquisa completa é impraticável. Nos sistemas de memória virtual, localizamos páginas usando uma tabela que indexa a memória; essa estrutura é chamada de **tabela de páginas** e reside na memória. Uma tabela de páginas é indexada pelo número de página do endereço virtual para descobrir o número da página física correspondente. Cada programa possui sua própria tabela de páginas, que mapeia o espaço de endereçamento virtual desse programa para a memória principal. Em nossa analogia da biblioteca, a tabela de páginas corresponde a um mapeamento entre os títulos dos livros e os locais da biblioteca. Exatamente como o catálogo de cartões pode conter entradas para livros em outra biblioteca ou campus em vez da biblioteca local, veremos que a tabela de páginas pode conter entradas para páginas não presentes na memória. A fim de indicar o local da tabela de páginas na memória, o hardware inclui um registrador que aponta para o início da tabela de páginas; esse registrador é chamado de *registrador de tabela de páginas*. Por enquanto, considere que a tabela de páginas esteja em uma área fixa e contígua da memória.

tabela de páginas A tabela com as traduções de endereço virtual para físico em um sistema de memória virtual. A tabela, armazenada na memória, normalmente é indexada pelo número de página virtual; cada entrada na tabela contém o número da página física para essa página virtual se a página estiver atualmente na memória.

Interface hardware/ software

A tabela de páginas, juntamente com o contador de programa e os registradores, especifica o *estado* de um programa. Se quisermos permitir que outro programa use o processador, precisamos salvar esse estado. Mais tarde, após restaurar esse estado, o programa pode continuar a execução. Frequentemente nos referimos a esse estado como um *processo*. O

processo é considerado *ativo* quando está de posse do processador; caso contrário, ele é considerado *inativo*. O sistema operacional pode tornar um processo ativo carregando o estado do processo, incluindo o contador de programa, o que irá iniciar a execução no valor salvo do contador de programa.

O espaço de endereçamento do processo, e, consequentemente, todos os dados que ele pode acessar na memória, é definido pela sua tabela de páginas, que reside na memória. Em vez de salvar a tabela de páginas inteira, o sistema operacional simplesmente carrega o registrador de tabela de páginas de modo a apontar para a tabela de páginas do processo que ele quer tornar ativo. Cada processo possui sua própria tabela de páginas, já que diferentes processos usam os mesmos endereços virtuais. O sistema operacional é responsável por alocar a memória física e atualizar as tabelas de páginas, de modo que os espaços de endereço virtuais dos diferentes processos não colidam. Como veremos em breve, o uso de tabelas de páginas separadas também fornece proteção de um processo contra outro.

A Figura 5.21 usa o registrador de tabela de páginas, o endereço virtual e a tabela de páginas indicada para mostrar como o hardware pode formar um endereço físico. Um bit de validade é usado em cada entrada de tabela de páginas, exatamente como faríamos em uma cache. Se o bit estiver desligado, a página não está presente na memória principal e ocorre uma falta de página. Se o bit estiver ligado, a página está na memória e a entrada contém o número de página física.

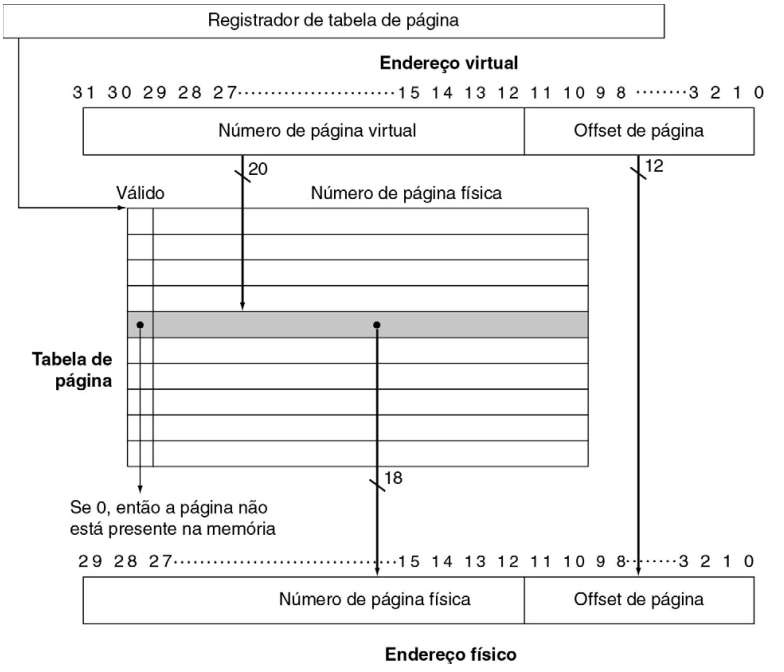

FIGURA 5.21 A tabela de páginas é indexada pelo número de página virtual para obter a parte correspondente do endereço físico. Consideramos um endereço de 32 bits. O endereço inicial da tabela de páginas é dado pelo ponteiro da tabela de páginas. Nessa figura, o tamanho de página é 2^{12} bytes, ou 4KB. O espaço de endereço virtual é 2^{32} bytes, ou 4GB, e o espaço de endereçamento físico é 2^{30} bytes, que permite uma memória principal de até 1GB. O número de entradas na tabela de páginas é 2^{20}, ou um milhão de entradas. O bit de validade para cada entrada indica se o mapeamento é legal. Se ele estiver desligado, a página não está presente na memória. Embora a entrada de tabela de páginas mostrada aqui só precise ter 19 bits de largura, ela normalmente seria arredondada para 32 bits a fim de facilitar a indexação. Os bits extras seriam usados para armazenar informações adicionais que precisam ser mantidas página a página, como a proteção.

Como a tabela de páginas contém um mapeamento para toda página virtual possível, nenhuma tag é necessária. Na terminologia da cache, o índice usado para acessar a tabela de páginas consiste no endereço de bloco inteiro, que é o número de página virtual.

Faltas de página

Se o bit de validade para uma página virtual estiver desligado, ocorre uma falta de página. O sistema operacional precisa receber o controle. Essa transferência é feita pelo mecanismo de exceção, que abordaremos posteriormente nesta seção. Quando o sistema operacional obtém o controle, ele precisa encontrar a página no próximo nível da hierarquia (geralmente o disco magnético) e decidir onde colocar a página requisitada na memória principal.

O endereço virtual por si só não diz imediatamente onde está a página no disco. Voltando à nossa analogia da biblioteca, não podemos encontrar o local de um livro nas estantes apenas sabendo seu título. Precisamos ir ao catálogo e consultar o livro, obter um endereço para o local nas estantes. Da mesma forma, em um sistema de memória virtual, é necessário monitorar o local no disco de cada página em um espaço de endereçamento virtual.

Como não sabemos de antemão quando uma página na memória será escolhida para ser substituída, o sistema operacional normalmente cria o espaço no disco para todas as páginas de um processo no momento em que ele cria o processo. Esse espaço do disco é chamado de **área de swap**. Nesse momento, o sistema operacional também cria uma estrutura para registrar onde cada página virtual está armazenada no disco. Essa estrutura de dados pode ser parte da tabela de páginas ou pode ser uma estrutura de dados auxiliar indexada da mesma maneira que a tabela de páginas. A Figura 5.22 mostra a organização quando uma única tabela contém o número de página física ou o endereço de disco.

área de swap O espaço no disco reservado para o espaço de memória virtual completo de um processo.

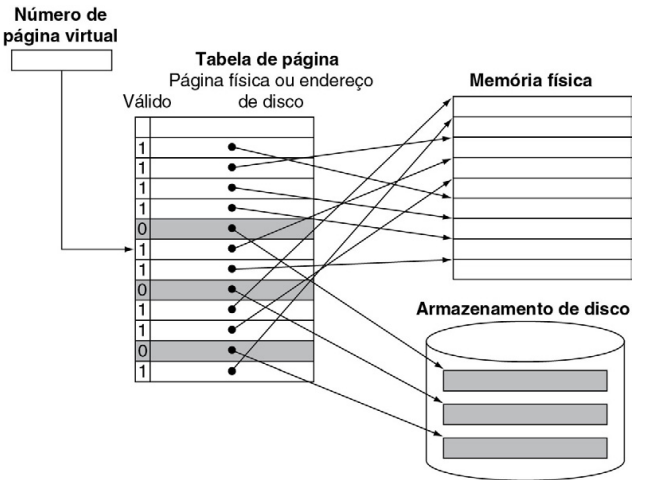

FIGURA 5.22 A tabela de páginas mapeia cada página na memória virtual em uma página na memória principal ou em uma página armazenada em disco, que é o próximo nível na hierarquia. O número de página virtual é usado para indexar a tabela de páginas. Se o bit de validade estiver ligado, a tabela de páginas fornece o número de página física (ou seja, o endereço inicial da página na memória) correspondente à página virtual. Se o bit de validade estiver desligado, a página reside atualmente apenas no disco, em um endereço de disco especificado. Em muitos sistemas, a tabela de endereços de página física e endereços de página de disco, embora sendo logicamente uma única tabela, é armazenada em duas estruturas de dados separadas. As tabelas duplas se justificam, em parte, porque precisamos manter os endereços de disco de todas as páginas, mesmo que elas estejam atualmente na memória principal. Lembre-se de que as páginas na memória principal e as páginas no disco são idênticas em tamanho.

O sistema operacional também cria uma estrutura de dados que controla quais processos e quais endereços virtuais usam cada página física. Quando ocorre uma falta de página, se todas as páginas na memória principal estiverem em uso, o sistema operacional precisa escolher uma página para substituir. Como queremos minimizar o número de faltas de página, a maioria dos sistemas operacionais tenta escolher uma página que supostamente não será necessária no futuro próximo. Usando o passado para prever o futuro, os sistemas operacionais seguem o esquema de substituição LRU (Least Recently Used – usado menos recentemente), que mencionamos na Seção 5.3. O sistema operacional procura a página usada menos recentemente, fazendo a suposição de que uma página que não foi usada por um longo período é menos provável de ser usada do que uma página acessada mais recentemente. As páginas substituídas são escritas na área de swap do disco. Caso você esteja curioso, o sistema operacional é apenas outro processo, e essas tabelas controlando a memória estão na memória; os detalhes dessa aparente contradição serão explicados em breve.

Interface hardware/ software

bit de referência Também chamado de **bit de uso**. Um campo que é ligado sempre que uma página é acessada e que é usado para implementar LRU ou outros esquemas de substituição.

Implementar um esquema de LRU completamente preciso é muito caro, pois requer atualizar uma estrutura de dados a *cada* referência à memória. Como alternativa, a maioria dos sistemas operacionais aproxima a LRU monitorando que páginas foram e que páginas não foram usadas recentemente. Para ajudar o sistema operacional a estimar as páginas LRU, alguns computadores fornecem um **bit de referência** ou **bit de uso**, que é ligado sempre que uma página é acessada. O sistema operacional limpa periodicamente os bits de referência e, depois, os registra para que ele possa determinar que páginas foram tocadas durante um determinado período. Com essas informações de uso, o sistema operacional pode selecionar uma página que está entre as referenciadas menos recentemente (detectadas tendo seu bit de referência desligado). Se esse bit não for fornecido pelo hardware, o sistema operacional precisará encontrar outra maneira de estimar que páginas foram acessadas.

Detalhamento: Com um endereço virtual de 32 bits, páginas de 4KB e 4 bytes por entrada da tabela de páginas, podemos calcular o tamanho total da tabela de páginas:

$$\text{Número de entradas da tabela de páginas} = \frac{2^{32}}{2^{12}} = 2^{20}$$

$$\text{Tamanho da tabela de páginas} = 2^{20} \text{ entradas da tabela de páginas} \; 2^{2} \frac{\text{bytes}}{\text{entrada da tabela de página}} = 4\,\text{MB}$$

Ou seja, precisaríamos usar 4MB da memória para cada programa em execução em um dado momento. Essa quantidade não é ruim para um único programa. Mas, e se houver centenas de programas rodando, cada um com sua própria tabela de página? E como devemos tratar endereços de 64 bits, que por esse cálculo precisaríamos de 2^{52} palavras?

Diversas técnicas são usadas no sentido de reduzir a quantidade de armazenamento necessária para a tabela de páginas. As cinco técnicas a seguir visam a reduzir o armazenamento máximo total necessário, bem como minimizar a memória principal dedicada às tabelas de páginas:

1. A técnica mais simples é manter um registrador de limite que restrinja o tamanho da tabela de páginas para um determinado processo. Se o número de página virtual se tornar maior do que o conteúdo do registrador de limite, entradas precisarão ser

incluídas na tabela de páginas. Essa técnica permite que a tabela de páginas cresça à medida que um processo consome mais espaço. Assim, a tabela de páginas só será maior se o processo estiver usando muitas páginas do espaço de endereçamento virtual. Essa técnica exige que o espaço de endereçamento se expanda apenas em uma direção.

2. Permitir o crescimento apenas em uma direção não é o bastante, já que a maioria das linguagens exige duas áreas cujo tamanho seja expansível: uma área contém a pilha e a outra contém o heap. Devido à essa dualidade, é conveniente dividir a tabela de páginas e deixá-la crescer do endereço mais alto para baixo, assim como do endereço mais baixo para cima. Isso significa que haverá duas tabelas de páginas separadas e dois limites separados. O uso de duas tabelas de páginas divide o espaço de endereçamento em dois segmentos. O bit mais significativo de um endereço normalmente determina que segmento – e, portanto, que tabela de páginas – deve ser usado para esse endereço. Como o segmento é especificado pelo bit de endereço mais significativo, cada segmento pode ter a metade do tamanho do espaço de endereçamento. Um registrador de limite para cada segmento especifica o tamanho atual do segmento, que cresce em unidades de páginas. Esse tipo de segmentação é usado por muitas arquiteturas, inclusive MIPS. Diferente do tipo de segmentação abordado na seção "Detalhamento" da Seção 5.4, essa forma de segmentação é invisível ao programa de aplicação, embora não para o sistema operacional. A principal desvantagem desse esquema é que ele não funciona bem quando o espaço de endereçamento é usado de uma maneira esparsa e não como um conjunto contíguo de endereços virtuais.

3. Outro método para reduzir o tamanho da tabela de páginas é aplicar uma função de hashing no endereço virtual de modo que a estrutura de dados da tabela de páginas precise ser apenas do tamanho do número de páginas *físicas* na memória principal. Essa estrutura é chamada de *tabela de páginas invertida*. É claro que o processo de consulta é um pouco mais complexo com uma tabela de páginas invertida porque não podemos mais simplesmente indexar a tabela de páginas.

4. Múltiplos níveis de tabelas de páginas também podem ser usados no sentido de reduzir a quantidade total de armazenamento para a tabela de páginas. O primeiro nível mapeia grandes blocos de tamanho fixo do espaço de endereçamento virtual, talvez de 64 a 256 páginas no total. Esses grandes blocos são, às vezes, chamados de segmentos, e essa tabela de mapeamento de primeiro nível é chamada de tabela de segmentos, embora os segmentos sejam invisíveis ao usuário. Cada entrada na tabela de segmentos indica se alguma página nesse segmento está alocada e, se estiver, aponta para uma tabela de páginas desse segmento. A tradução de endereços ocorre primeiramente olhando na tabela de segmentos, usando os bits mais significativos do endereço. Se o endereço do segmento for válido, o próximo conjunto de bits mais significativos é usado para indexar a tabela de páginas indicada pela entrada da tabela de segmentos. Esse esquema permite que o espaço de endereçamento seja usado de uma maneira esparsa (vários segmentos não contíguos podem estar ativos), sem precisar alocar a tabela de páginas inteira. Esses esquemas são particularmente úteis com espaços de endereçamento muito grandes e em sistemas de software que exigem alocação não contígua. A principal desvantagem desse mapeamento de dois níveis é o processo mais complexo para a tradução de endereços.

5. A fim de reduzir a memória principal real consumida pelas tabelas de páginas, a maioria dos sistemas modernos também permite que as tabelas de páginas sejam paginadas. Embora isso pareça complicado, esse esquema funciona usando os mesmos conceitos básicos da memória virtual e simplesmente permite que as tabelas de páginas residam no espaço de endereçamento virtual. Entretanto, há alguns problemas pequenos mas cruciais, como uma série interminável de faltas de página, que precisam ser evitadas. A forma como esses problemas são resolvidos é um tema muito detalhado e, em geral, altamente específico ao processador. Em poucas palavras, esses problemas são evitados colocando todas as tabelas de páginas no espaço de endereçamento do

sistema operacional e colocando pelo menos algumas das tabelas de páginas para o sistema em uma parte da memória principal que é fisicamente endereçada e está sempre presente – e, portanto, nunca no disco.

E quanto às escritas?

A diferença entre o tempo de acesso à cache e à memória principal é de dezenas a centenas de ciclos, e os esquemas write-through podem ser usados, embora precisemos de um buffer de escrita para ocultar do processador a latência da escrita. Em um sistema de memória virtual, as escritas no próximo nível de hierarquia (disco) levam milhões de ciclos de clock de processador; portanto, construir um buffer de escrita para permitir que o sistema escreva diretamente no disco seria impraticável. Em vez disso, os sistemas de memória virtual precisam usar write-back, realizando as escritas individuais para a página na memória e copiando a página novamente para o disco quando ela é substituída na memória.

Interface hardware/ software

Um esquema write-back possui outra importante vantagem em um sistema de memória virtual. Como o tempo de transferência de disco é pequeno comparado com seu tempo de acesso, copiar de volta uma página inteira é muito mais eficiente do que escrever palavras individuais novamente no disco. Uma operação write-back, embora mais eficiente do que transferir páginas individuais, ainda é onerosa. Portanto, gostaríamos de saber se uma página *precisa* ser copiada de volta quando escolhemos substituí-la. Para monitorar se uma página foi escrita desde que foi lida para a memória, um *bit de modificação* (dirty bit) é acrescentado à tabela de páginas. O bit de modificação é ligado quando qualquer palavra em uma página é escrita. Se o sistema operacional escolher substituir a página, o bit de modificação indica se a página precisa ser escrita no disco antes que seu local na memória possa ser cedido a outra página. Logo, uma página modificada normalmente é chamada de "dirty page".

Tornando a tradução de endereços rápida: a TLB

Como as tabelas de páginas são armazenadas na memória principal, cada acesso à memória por um programa pode levar, no mínimo, o dobro do tempo: um acesso à memória para obter o endereço físico e um segundo acesso para obter os dados. O segredo para melhorar o desempenho de acesso é basear-se na localidade da referência à tabela de páginas. Quando uma tradução para um número de página virtual é usada, ela provavelmente será necessária novamente no futuro próximo, pois as referências às palavras nessa página possuem localidade temporal e também espacial.

Assim, os processadores modernos incluem uma cache especial que controla as traduções usadas recentemente. Essa cache especial de tradução de endereços é tradicionalmente chamada de **TLB (translation-lookaside buffer)**, embora seria mais correto chamá-la de cache de tradução. A TLB corresponde àquele pequeno pedaço de papel que normalmente usamos para registrar o local de um conjunto de livros que consultamos no catálogo; em vez de pesquisar continuamente o catálogo inteiro, registramos o local de vários livros e usamos o pedaço de papel como uma cache da biblioteca.

TLB (Translation-Lookaside Buffer) Uma cache que monitora os mapeamentos de endereços recentemente usados para evitar um acesso à tabela de páginas.

A Figura 5.23 mostra que cada entrada de tag na TLB contém uma parte do número de página virtual, e cada entrada de dados da TLB contém um número de página física. Como não iremos mais acessar a tabela de páginas a cada referência, em vez disso acessaremos a TLB, que precisará incluir outros bits de status, como o bit de modificação e o bit de referência.

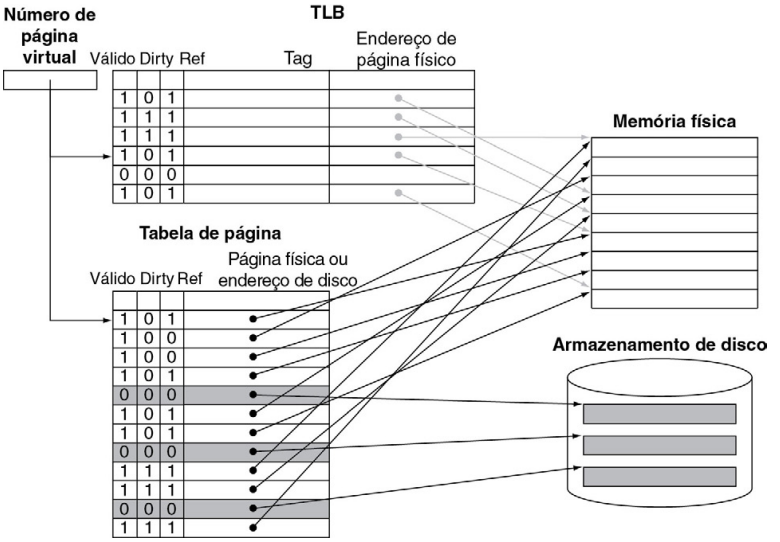

FIGURA 5.23 A TLB age como uma cache da tabela de páginas apenas para as entradas que mapeiam as páginas físicas. A TLB contém um subconjunto dos mapeamentos de página virtual para física que estão na tabela de páginas. Os mapeamentos da TLB são mostrados em destaque. Como a TLB é uma cache, ela precisa ter um campo tag. Se não houver uma entrada correspondente na TLB para uma página, a tabela de páginas precisa ser examinada. A tabela de páginas fornece um número de página física para a página (que pode, então, ser usado na construção de uma entrada da TLB) ou indica que a página reside em disco, caso em que ocorre uma falta de página. Como a tabela de páginas possui uma entrada para cada página virtual, nenhum campo tag é necessário; ou seja, ela *não* é uma cache.

Em cada referência, consultamos o número de página virtual na TLB. Se tivermos um acerto, o número de página física é usado para formar o endereço e o bit de referência correspondente é ligado. Se o processador estiver realizando uma escrita, o bit de modificação também é ligado. Se ocorrer uma falha na TLB, precisamos determinar se ela é uma falta de página ou simplesmente uma falha de TLB. Se a página existir na memória, então a falha de TLB indica apenas que a tradução está faltando. Nesse caso, o processador pode tratar a falha de TLB lendo a tradução da tabela de páginas para a TLB e, depois, tentando a referência novamente. Se a página não estiver presente na memória, então a falha de TLB indica uma falta de página verdadeira. Nesse caso, o processador chama o sistema operacional usando uma exceção. Como a TLB possui muito menos entradas do que o número de páginas na memória principal, as falhas de TLB serão muito mais frequentes do que as faltas de página verdadeiras.

As falhas de TLB podem ser tratadas no hardware ou no software. Na prática, com cuidado, pode haver pouca diferença de desempenho entre os dois métodos, uma vez que as operações básicas são iguais nos dois casos.

Depois que uma falha de TLB tiver ocorrido e a tradução faltando tiver sido recuperada da tabela de páginas, precisaremos selecionar uma entrada da TLB para substituir. Como os bits de referência e de modificação estão contidos na entrada da TLB, precisamos copiar esses bits de volta para a entrada da tabela de páginas quando substituirmos uma entrada. Esses bits são a única parte da entrada da TLB que pode ser modificada. O uso de write-back – ou seja, copiar de volta essas entradas no momento da falha e não quando são escritas – é muito eficiente, já que esperamos que a taxa de falhas da TLB seja pequena. Alguns sistemas usam outras técnicas para aproximar os bits de referência e de modificação, eliminando a necessidade de escrever na TLB exceto para carregar uma nova entrada da tabela em caso de falha.

Alguns valores comuns para uma TLB poderiam ser:

- Tamanho da TLB: 16 a 512 entradas.
- Tamanho do bloco: uma a duas entradas da tabela de páginas (geralmente 4 a 8 bytes cada uma).

David A. Patterson
John L. Hennessy

ORGANIZAÇÃO E PROJETO DE COMPUTADORES

A INTERFACE HARDWARE/SOFTWARE

Tradução da 4ª Edição

ELSEVIER
CAMPUS