# **Capítulo 8**

**Main Memory**

Background

→ O programa deve ser trazido (do disco) para a memória e colocado dentro de um processo para que seja executado

→ Memória principal e registos são apenas armazenamento que a CPU pode aceder diretamente

→ Unidade de memória só vê um fluxo de registos + pedidos de leitura, ou registo + dados e pedidos de escrita

→ Acesso a registos em um CPU clock (ou menos)

→ Memória principal pode demorar muitos ciclos, causando uma stall

→ Cache fica entre a memória principal e os registos da CPU

→ Proteção da memória necessária para garantir o correto funcionamento

Base and Limit Registers

→ Um par de registos base e limite definem o espaço de endereçamento lógico

→ CPU deve verificar todos os acessos de memória gerados no modo user para ter a certeza de que está entre a base e o limite para esse utilizador

Address Binding

→ Programas em disco, prontos para serem trazidos para a memória para execução de uma input queue

* Sem suporte, deve ser carregado no registo 0000

→ Inconveniente para ter o primeiro endereço físico do processo do utilizador sempre em 0000

→ Além disso, registos representados de diferentes maneiras em diferentes fases da vida

* do código fonte de um programa são registos geralmente simbólicos
* de registos de código compilado de um programa ligam relocatable addresses
* do linker ou do loader de um programa ligam relocatable addresses a registos absolutos
* de cada mapa de ligação de um programa de um espaço de endereçamento para outro

Binding of Instructions and Data to Memory

→ A ligação das instruções e dados aos registos de memória pode acontecer em três fases diferentes

* Compile Time
* Se a localização da memória for conhecida a priori, o absolute code pode ser gerado; deve recompilar o código se a localização começar a mudar
* Load Time
* Deve gerar relocatable code se a localização da memória não for conhecida no compile time
* Execution Time
* Ligação adiada até ao tempo de funcionamento se o processo puder ser movido durante a sua execução de um segmento de memória para outro

Necessário suporte de hardware para mapas de registos (ex: registos de base e limite)

Logical vs. Physical Address Space

→ O conceito de um espaço de endereçamento lógico que está ligado a um espaço de endereçamento físico separado é central para a gestão adequada da memória

* Endereço Lógico: gerado pela CPU; também referido como endereço virtual
* Endereço Físico: endereço visto pela unidade de memória

→ Endereços lógicos e físicos são o mesmo em esquemas de ligação ao endereço em tempo de compilação e carregamento; endereços lógicos (virtuais) e físicos diferem no esquema de execução-tempo de ligação de endereços

→ Logical address space é o conjunto de todos os endereços lógicos gerados por um programa

→ Physical address space é o conjunto de todos os endereços físicos gerados por um programa

**Memory-Management Unit (MMU)**

→ Dispositivo de hardware que no tempo de execução mapeia o endereço virtual para o físico

→ Para começar, considerar um esquema simples onde o valor no relocatable address é adicionado a cada endereço gerado por um processo de utilizador no momento em que é enviado para a memória

* base de memória agora chamado relocation register
* MS-DOS na Intel 80x86 utilizou 4 relocatable addresses

→ O programa do utilizador trata de endereços lógicos; nunca vê os endereços físicos reais

* A ligação em tempo de execução ocorre quando é feita referência à localização na memória
* Endereço lógico ligado a endereços físicos

Dynamic relocation using a relocation register

→ A rotina não é carregada até que seja chamada

→ Melhor utilização do espaço de memória; rotina não suportada nunca é carregada

→ Todas as rotinas mantidas no disco em formato de relocatable load

→ Útil quando grandes quantidades de código são necessárias para lidar com casos pouco frequentes

→ Não é necessário um suporte especial do SO

* Implementado através do design do programa
* SO pode ajudar fornecendo bibliotecas para implementar dynamic loading

**Dynamic Linking**

→ Definição: ligação adiada até tempo de execução

→ Static Linking: bibliotecas do sistema e código do programa combinado pelo loader na imagem do programa binário

→ Pequeno pedaço de código, stub, usado para localizar a rotina adequada da biblioteca residente em memória

→ O Stub substitui-se pelo endereço da rotina, e executa a rotina

→ SO verifica se a rotina estiver no endereço de memória dos processos

* Se não estiver no espaço de endereçamento, adicionar para o espaço

→ Ligação dinâmica é particularmente útil para bibliotecas

→ Sistema também conhecido como shared libraries

→ Considera a aplicabilidade para corrigir bibliotecas do sistema

* Versionamento (processo de atribuir nome ou numeração única para indicar o estado de um programa) pode ser necessário

**Swapping**

→ Um processo pode ser trocado temporariamente da memória para uma memória secundária, e depois trazido de volta à memória para execução contínua

* O espaço de memória física total dos processos pode exceder a memória física

→ Memória secundária

* Disco rápido suficientemente grande para acomodar cópias de todas as imagens de memória para todos os utilizadores; deve fornecer acesso direto a estas imagens de memória

→ Roll out, roll in

* Variante de troca utilizada para algoritmos de agendamento baseados em prioridades; O processo de menor prioridade é trocado para que o processo de maior prioridade possa ser carregado e executado

→ A maior parte do tempo de swap é o tempo de transferência; o tempo total de transferência é diretamente proporcional à quantidade de memória trocada

→ O Sistema mantém uma ready queue de processos prontos a executar que têm imagens de memória no disco

→ O processo de swap precisa de ser trocado para os mesmos endereços físicos?

→ Depende do método de ligação ao endereço

* Além disso, considerar I/O pendente para/do espaço de memória de processo

→ As versões modificadas de troca são encontradas em muitos sistemas (ex: UNIX, Linux e Windows)

* Troca normalmente desativada
* Iniciada se mais do que a quantidade limite de memória atribuída
* Desativada uma vez que a procura de memória reduzida abaixo do limite

Context Switch Time Including Swapping

→ Se os próximos processos a colocar no CPU não estiverem na memória, precisa trocar um processo e trocar em processo-alvo

→ O tempo de context switch de 100MB pode ser muito elevado

→ Troca de processo de 100MB para o disco rígido com taxa de transferência de 50MB/seg

* Troca de tempo de 2000ms
* Troca de processo do mesmo tamanho
* Tempo total do componente de troca de contexto switch de 4000ms (4 segundos)

→ Pode reduzir se reduzir o tempo de troca – sabendo a quantidade de memória que realmente está a ser usada

* System calls informam o SO do uso da memória através de request\_memory() e release\_memory()

→ Outras restrições também na troca

* I/O pendente - não podem ser trocados porque o I/O ocorreria no processo errado
* Ou transferir sempre I/O para o espaço do kernel, depois para o dispositivo de I/O
* Conhecido como double buffering, adiciona troca
* Padrão geral de troca não utilizada em SO modernos
* Mas versão modificada comum

Troca apenas quando a memória livre é extremamente baixa

Swapping em Sistemas Móveis

→ Normalmente não suporta

* Memória flash com base em
* Pequena quantidade de espaço
* Número limitado de ciclos de escrita
* Fraca produção entre memória flash e CPU na plataforma móvel

→ Em vez disso, usa outros métodos para libertar a memória se

* o iOS pedir às aplicações para libertarem voluntariamente a memória alocada
* Dados de leitura apenas deitados fora e recarregados da flash se necessário
* Falha para libertar pode resultar em terminação
* Android termina aplicações se a memória livre for baixa, mas primeiro escreve o estado de aplicação para a flash para reiniciar rapidamente
* Ambos os SO suportarem paginação

**Alocação Contígua**

→ A memória principal deve suportar tanto os processos de SO como os processos do utilizador

→ Recursos limitados, deve atribuir eficientemente

→ A alocação contígua é um método inicial

→ Memória principal geralmente em duas partitions:

* SO residente, geralmente mantido em memória baixa com interrupt vector
* Processos do utilizador interrompidos são mantidos em alta memória
* Cada processo contido numa única secção contígua de memória

→ Relocation registers utilizados para proteger os processos dos utilizadores uns dos outros, e da alteração do código e dos dados do SO

* Registo base contém o valor do menor registo de registos físicos
* O registo limite contém uma gama de endereços lógicos – cada endereço lógico deve ser inferior ao endereço lógico do registo limite
* MMU mapeia o endereço lógico dinamicamente
* Pode depois permitir que ações como o código kernel sejam transient e o tamanho do kernel mudando

**Alocação de múltiplas partições**

→ Grau de multiprogramação limitado pelo número de partições

→ Tamanhos de partição variáveis para eficiência (dimensionado a um determinado processo)

→ Hole – bloco de memória disponível; holes de vários tamanhos são espalhados por toda a memória

→ Quando um processo chega, é alocada memória de um hole grande o suficiente para acomodá-lo

→ O processo de saída liberta a sua partição, divisórias livres adjacentes combinadas

→ SO mantém informações sobre:

* partições alocadas
* partições livres (furo)

**Problema dinâmico de alocação de armazenamento**

→ Como satisfazer um pedido de tamanho n de uma lista de holes livres?

* First-fit: Alocar o primeiro hole suficientemente grande
* Best-fit: Alocar o hole mais pequeno que seja grande o suficiente; deve pesquisar toda a lista, a menos que ordenado por tamanho
* Produz o menor hole restante
* Worst-fit: Alocar o maior hole; deve também pesquisar a lista inteira
* Produz o maior hole que sobra

→ Firt-fit e best-fit são melhores do que o worst-fit em termos de utilização de velocidade e armazenamento

**Fragmentação**

→ Fragmentação Externa: espaço de memória total existe para satisfazer um pedido, mas não é contígua

→ Fragmentação Interna: a memória alocada pode ser ligeiramente maior do que a memória solicitada; esta diferença de tamanho é a memória interna para uma partição, mas não sendo usada

→ A análise do first-fit revela que dado blocos N atribuídos, blocos de 0,5 N perdidos para fragmentação

* 1/3 pode ser inutilizável (50-percent rule)

→ Reduzir a fragmentação externa por compactação

* Baralhar os conteúdos da memória para colocar toda a memória livre junta num grande bloco
* Compactação só é possível se a relocation for dinâmica, e for feita no tempo de execução
* Problema I/O
* Trabalho da latch na memória enquanto está envolvido em I/O
* Fazer I/O apenas em buffers do SO

→ Agora considere que a memória secundária tem os mesmos problemas de fragmentação

**Segmentação**

Definição

→ Esquema da gestão de memória que suporta a visão do utilizador da memória

→ Um programa é uma coleção de segmentos

* Um segmento é uma unidade lógica, tal como
* programa principal, procedimento, função, método, objeto, variáveis locais e globais, bloco comum, pilha, tabela de símbolos, arrays

Arquitetura

→ O endereço lógico consiste em duas tuplas

→ Segment table: mapeia endereços físicos bidimensionais; cada entrada de tabela tem

* base – contém o endereço físico inicial onde os segmentos residem na memória
* limite – especifica o comprimento do segmento

→ Segment-table base register (STBR) aponta para a localização da tabela de segmentos

→ Segment-table length register (STLR) indica o número de segmentos utilizados por um programa

* número de segmento s é legal se s < STLR

→ Proteção

* Com cada entrada na tabela de segmento associada
* bit de validação = 0 ⇒ segmento ilegal
* ler/escrever/executar privilégios

→ Bits de proteção associados a segmentos; partilha de código ocorre ao nível do segmento

→ Uma vez que os segmentos variam de comprimento, a alocação de memória é um problema dinâmico de atribuição de armazenamento

Vantagens

→ Processos divididos em segmentos de forma natural de acordo com função

→ Associar permissões de acesso por segmento

→ Cada segmento usa apenas/estritamente espaço de que necessita

Desvantagens

→ Segmentos de tamanho diferente dificultam gestão da memória

→ Algoritmos de inserção/substituição segmento mais complicados

**Paginação**

Definição

→ Espaço de endereçamento físico de um processo pode ser não contíguo; processo é alocado na memória física sempre que este último está disponível

* Evita a fragmentação externa
* Evita o problema de diferentes pedaços de memória de tamanho variado

→ Divide a memória física em blocos de tamanho fixo chamados frames

* Tamanho é potência de 2, entre 512 bytes e 16 Mbytes

→ Divide a memória lógica em blocos de páginas de tamanho igual chamadas pages

→ Mantém o registo de todos os frames livres

→ Para executar um programa de páginas de tamanho N, precisa de encontrar N frames livres e o programa de load

→ Configura uma page table para traduzir endereços lógicos para físicos

→ A memória secundária também é dividida em páginas

→ Ainda tem fragmentação interna

→ Calcular fragmentação interna

* Tamanho página = 2.048 bytes
* Tamanho processo = 72.766 bytes
* 35 páginas + 1.086 bytes
* Fragmentação interna de 2.048 - 1.086 = 962 bytes
* Pior fragmento do caso = 1 frame – 1 byte
* Em média fragmentação = tamanho frame 1 / 2 tamanho frame
* Tamanhos tão pequenos dos frames são desejáveis?
* Mas cada entrada na page table requer memória para rastrear
* Tamanhos de página crescendo ao longo do tempo
* Solaris suporta dois tamanhos de página - 8 KB e 4 MB

→ Visão de processo e memória física agora muito diferente

→ Por processo de implementação só pode aceder à sua própria memória

Address Translation Scheme

→ O endereço gerado pela CPU é dividido em

* Page number (p): usado como um índice numa page table que contém endereço base de cada página na memória física
* Page Offset (d): combinado com endereço base para definir o endereço de memória física enviado para a unidade de memória
* Para dado espaço de endereçamento lógico 2^m e tamanho da página 2^n

Implementação da page table

→ Page table é mantida na memória principal

→ Page-table base register (PTBR) aponta para a page table

→ Page-table length register (PTLR) indica o tamanho da page table

→ Neste esquema, cada acesso de dados/instruções requer dois acessos à memória

* Um para a page table e um para dados/instruções

→ O problema de acesso às duas memórias pode ser resolvido através da utilização de um hardware especial de procura rápida chamado associative memory ou translation look-aside buffers (TLBs)

→ Alguns address-space identifiers (ASIDs) em cada entrada TLB – identifica unicamente cada processo para fornecer proteção address-space para esse processo

* De outra forma, precisa de descarregar em cada context switch

→ TLBs normalmente pequenas (64 a 1024 entradas)

→ Numa falha de TLB, o valor é carregado no TLB para um acesso mais rápido da próxima vez que

* as políticas de substituição devam ser consideradas
* algumas entradas possam ser weird down para acesso rápido permanente

**Memória Associativa**

→ Definição: pesquisa paralela

→ Address translation (p, d)

* Se p estiver no registo associativo, get frame # out
* De outra forma get frame # da page table na memória

**Effective Access Time**

→ Associative Lookup = ε time unit

* Pode ser < 10% do tempo de acesso à memória

→ Hit ratio = α

* Hit ratio – percentagem de vezes que um número de página é encontrado nos registos associativos; ratio relacionado com o número de registos associativos

→ Considere α = 80%, ε = 20ns para pesquisa de TLB, 100ns para acesso à memória

→ Effective Access Time (EAT)

* EAT = (1 + ε) α + (2 + ε )(1 – α) = 2 + ε – α

→ Considere α = 80%, ε = 20ns para pesquisa de TLB, 100ns para acesso à memória

* EAT = 0,80 x 100 + 0,20 x 200 = 120ns

→ Considere uma relação de impacto mais realista -> α = 99%, ε = 20ns para pesquisa de TLB, 100ns para acesso à memória

* EAT = 0,99 x 100 + 0,01 x 200 = 101ns

**Proteção da memória**

→ Implementada associando a parte de proteção a cada frame para indicar se é permitido o acesso apenas de leitura ou leitura-escrita

→ Pode também adicionar mais bits para indicar apenas a execução da página…

→ Valid-invalid bit anexado a cada entrada na page table:

* "valid" indica que a página associada está no espaço de endereçamento lógico do processo e é, portanto, uma página legal
* "invalid" indica que a página não está no espaço de endereçamento lógico do processo
* Ou usa o registo page-table length register (PTLR)

→ Quaisquer violações resultam numa armadilha para o núcleo

**Shared Pages**

Shared code

→ Uma cópia do código apenas de leitura (reentrant) partilhada entre processos (isto é, editores de texto, compiladores, sistemas de janelas)

→ Semelhante a múltiplos fios que partilham o mesmo espaço de processo

→ Também útil para comunicação interprocessada se a partilha de páginas de leitura-escrita é permitida

Private code and data

→ Cada processo mantém uma cópia separada do código e dados

→ As páginas do código e dados privados podem aparecer em qualquer lugar no espaço do endereço lógico

**Estrutura Page Table**

→ Estruturas da memória da page table para paginação podem ficar enormes usando métodos simples

* Considerar um espaço de endereçamento lógico de 32 bits como em computadores modernos
* Tamanho de página de 4 KB (2^12)
* Tabela de página teria 1 milhão de entradas (2^32 / 2^12)
* Se cada entrada for de 4 bytes -> 4 MB de espaço de endereçamento físico/memória apenas para a page table
* Essa quantidade de memória usada costumava custar muito
* Não se quer alocar isso contiguamente na memória principal

→ Hierarchical Paging

→ Hashed Page Tables

* Comuns em espaços de endereço > 32 bits
* O número de página virtual é hashed numa page table
* Esta tabela de página contém uma cadeia de elementos hashing para o mesmo local
* Cada elemento contém (1) o número de página virtual (2) o valor do frame da página mapeada (3) um ponteiro para o elemento seguinte
* Os números da página virtual são comparados nesta cadeia procurando por uma correspondência
* Se for encontrado uma correspondência, o frame físico correspondente é extraído
* Variação para endereços de 64 bits é page tables agrupadas
* Semelhantes a hashed, mas cada entrada refere-se a várias páginas (tais como 16) em vez de 1
* Especialmente útil para espaços de endereçamento escassos (onde as referências de memória não são contíguas e dispersas)

→ Inverted Page Tables

* Em vez de cada processo ter uma page table e acompanhar todas as páginas lógicas possíveis, rastrear todas as páginas físicas
* Uma entrada para cada página real de memória
* A entrada consiste no endereço virtual da página armazenada nesse local de memória real, com informações sobre o processo que possui essa página
* Diminuir a memória necessária para armazenar cada tabela de página, mas aumenta o tempo necessário para pesquisar a tabela quando ocorre uma referência de página
* Usar hash table para limitar a pesquisa a uma (ou no máximo algumas) entradas de page table
* TLB pode acelerar o acesso
* Mas como implementar memória partilhada?
* Um mapeamento de um endereço virtual para o endereço físico compartilhado

→ Hierarchical Page Tables

* Partir o espaço de endereçamento lógico em várias page tables
* Uma técnica simples é uma page table de dois níveis
* Depois paginamos a page table