Capítulo 1

Introdução

1.1 Arquitetura IA-32/x86-64

A arquitetura Intel x86 representa um dos conjuntos de instruções mais importantes e duradouros na história da computação moderna. Desde sua introdução com o processador 8086 em 1978, a arquitetura evoluiu substancialmente, mantendo compatibilidade retroativa enquanto incorporava novas capacidades e extensões. Esta seção explora os fundamentos da arquitetura x86, com foco nas suas características relevantes para o gerenciamento de memória.

1.1.1 Evolução da Arquitetura x86

A arquitetura x86 passou por transformações significativas ao longo das décadas:

- **8086/8088 (1978)**: Processador de 16 bits com barramento de endereço de 20 bits, permitindo acessar 1MB de memória através de um sistema de segmentação.
- **80286 (1982)**: Introduziu o modo protegido, expandindo o espaço de endereçamento para 16MB e adicionando proteção de memória mais robusta.
- **80386 (1985)**: Primeiro processador x86 de 32 bits (IA-32), com barramento de endereço de 32 bits permitindo acessar 4GB de memória física. Introduziu a paginação, fundamental para sistemas operacionais modernos.
- **80486 e Pentium (1989-1993)**: Refinamentos da arquitetura IA-32 com melhorias de desempenho, incluindo cache L1 integrado.
- **AMD64/Intel 64 (2003)**: Extensão de 64 bits da arquitetura x86, permitindo endereçar teoricamente até 2^{64} bytes de memória (limitado na prática a 48 ou 57 bits de endereçamento físico nos processadores atuais).

1.1.2 Modos de Operação

A arquitetura x86 moderna suporta múltiplos modos de operação:

- **Modo Real**: Emula o ambiente do 8086 original, com endereçamento segmentado de 16 bits. Limitado a 1MB de memória sem proteção entre programas.
- **Modo Protegido**: Introduzido no 80286, oferece proteção de memória através de segmentação e, a partir do 80386, paginação. Este é o modo utilizado pelos sistemas operacionais de 32 bits.
- **Modo Virtual 8086**: Permite executar programas de modo real dentro do ambiente protegido de multitarefa.

- **Modo Longo (64 bits)**: Disponível em processadores AMD64/Intel 64, oferece espaço de endereçamento de 64 bits, registradores estendidos e outros recursos avançados.
- **SMM (System Management Mode)**: Modo especial usado para funções de gerenciamento do sistema, como controle de energia.

1.1.3 Registradores do Processador

Os registradores são componentes fundamentais para o funcionamento da CPU. Na arquitetura x86, dividem-se em várias categorias:

- **Registradores de Propósito Geral**: Em modo de 32 bits, incluem EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP. Em modo de 64 bits, são estendidos para RAX, RBX, etc., e novos registradores (R8-R15) são adicionados.
- **Registradores de Segmento**: CS (Code Segment), DS (Data Segment), SS (Stack Segment), ES, FS, GS. Fundamentais para o modelo de segmentação.
- **Registradores de Controle**: CR0-CR4 (em IA-32) e CR0-CR8 (em x86-64), controlam aspectos do processador como paginação, cache e modos de operação.
- **Registradores de Sistema**: GDTR, LDTR, IDTR e TR, apontam para estruturas do sistema como GDT (Global Descriptor Table), LDT (Local Descriptor Table), IDT (Interrupt Descriptor Table) e TSS (Task State Segment).
- **Registradores de Depuração**: DR0-DR7, utilizados para depuração de hardware.
- **Registradores de MMX, SSE, AVX**: Suportam instruções SIMD (Single Instruction Multiple Data) para processamento vetorial.

1.1.4 Proteção e Privilégios

A arquitetura x86 implementa um sistema de proteção baseado em anéis de privilégios, numerados de 0 (mais privilegiado) a 3 (menos privilegiado):

- **Anel 0**: Reservado para o núcleo do sistema operacional (kernel). Tem acesso irrestrito ao hardware e pode executar todas as instruções.
- **Anel 1 e 2**: Raramente utilizados nos sistemas operacionais modernos, foram projetados para drivers de dispositivos e serviços do sistema.
- **Anel 3**: Onde executam as aplicações do usuário, com acesso restrito ao hardware e impossibilitadas de executar instruções privilegiadas.

Esta hierarquia de privilégios é fundamental para a segurança do sistema, impedindo que programas de usuário acessem diretamente o hardware ou interfiram com outros programas.

1.1.5 Gerenciamento de Interrupções e Exceções

O processador x86 possui um sistema sofisticado para lidar com interrupções e exceções:

- **Interrupções**: Podem ser geradas externamente (hardware) ou internamente (software via instrução INT).
- **Exceções**: Ocorrem quando o processador detecta uma condição anormal durante a execução de uma instrução, como divisão por zero ou falha de página.
- **IDT (Interrupt Descriptor Table)**: Tabela que associa vetores de interrupção/exceção a rotinas de tratamento.

Entre as exceções mais relevantes para o gerenciamento de memória estão:

- **Page Fault (14)**: Ocorre quando um programa tenta acessar uma página não presente na memória física ou sem permissões adequadas.
- **General Protection Fault (13)**: Ocorre por violações de proteção, como tentar executar uma instrução privilegiada em modo usuário ou acessar memória além dos limites do segmento.

1.1.6 Conceitos de Memória Baremetal

Programação baremetal refere-se ao desenvolvimento de software que executa diretamente no hardware sem um sistema operacional subjacente. No contexto da arquitetura x86, isso envolve:

- **Inicialização**: O processador inicia em Modo Real e deve ser configurado para Modo Protegido ou Longo pelo código de inicialização.
- **GDT e IDT**: O código de inicialização deve configurar estas estruturas fundamentais.
- **Paginação**: Se desejada, deve ser habilitada explicitamente.
- **Interação direta com hardware**: O código tem acesso direto a todas as instruções e portas de E/S.

A programação baremetal é valiosa educacionalmente pois expõe os mecanismos internos do processador normalmente abstraídos pelos sistemas operacionais, permitindo uma compreensão mais profunda da arquitetura.

1.2 Gerenciamento de Memória na Arquitetura x86

O sistema de gerenciamento de memória na arquitetura x86 evoluiu significativamente ao longo do tempo, implementando mecanismos cada vez mais sofisticados para atender às crescentes demandas por espaço de endereçamento, proteção e eficiência. Esta seção explora os principais componentes e conceitos do sistema de memória na arquitetura Intel.

1.2.1 Modos de Endereçamento

A arquitetura x86 possui diferentes modos de endereçamento que evoluíram com o tempo:

- **Endereçamento em Modo Real**: Usa um esquema de segmentação simples onde o endereço físico é calculado multiplicando o valor do registro de segmento por 16 (deslocamento de 4 bits) e adicionando o deslocamento (offset). Limita-se a 1MB (2^{20} bytes) de memória endereçável.
- **Endereçamento em Modo Protegido (32 bits)**: Suporta até 4GB (2^{32} bytes) de memória física e implementa mecanismos de proteção através de segmentação avançada e paginação.
- **Endereçamento em Modo Longo (64 bits)**: Expande o espaço de endereçamento para teoricamente 2^{64} bytes, embora implementações atuais suportem 48 ou 57 bits de endereçamento. A segmentação é simplificada neste modo, com foco principal na paginação.

1.2.2 Segmentação

A segmentação é um dos mecanismos de gerenciamento de memória fundamentais na arquitetura x86, especialmente em modos Real e Protegido.

Segmentação em Modo Protegido

No modo protegido, a segmentação torna-se mais sofisticada:

- **Descritores de Segmento**: Estruturas de 8 bytes que definem as propriedades de cada segmento, incluindo:
  - Base (endereço inicial do segmento na memória física)
  - Limite (tamanho do segmento)
  - Tipo (código, dados, sistema)
  - Privilégios (nível de privilégio exigido para acesso)

  - Flags diversos (presença, granularidade, etc.)

- **Tabelas de Descritores**: Os descritores são organizados em tabelas:
  - **GDT (Global Descriptor Table)**: Tabela global de descritores compartilhada por todos os processos
  - **LDT (Local Descriptor Table)**: Tabela local específica para um processo ou grupo de processos
- **Seletores de Segmento**: Valores de 16 bits carregados nos registradores de segmento (CS, DS, SS, ES, FS, GS), composto por:
  - Índice (bits 3-15): seleciona um descritor na tabela
  - TI (bit 2): indica se o descritor está na GDT (0) ou LDT (1)
  - RPL (bits 0-1): Nível de privilégio solicitado

Proteção via Segmentação

A segmentação fornece vários mecanismos de proteção:

- **Verificação de Limites**: Previne acesso fora dos limites definidos para o segmento
- **Níveis de Privilégio**: Restringe acesso com base nos níveis de privilégio (anéis 0-3)
- **Controle de Tipo**: Previne execução de dados ou escrita em código
- **Verificação de Presença**: Garante que o segmento está presente na memória física

1.2.3 Paginação

A paginação é o mecanismo mais importante para gerenciamento de memória em sistemas operacionais modernos baseados em x86. Ao contrário da segmentação que opera com segmentos de tamanhos variáveis, a paginação divide a memória em unidades de tamanho fixo chamadas páginas (tipicamente 4KB).

Características Principais

- **Tradução de Endereços**: Converte endereços lineares (pós-segmentação) em endereços físicos
- **Granularidade Fina**: Opera com unidades fixas (páginas), facilitando a alocação e gerenciamento

- **Memória Virtual**: Permite que o espaço de endereçamento virtual exceda a memória física disponível
- **Compartilhamento**: Múltiplos processos podem mapear a mesma página física
- **Proteção por Página**: Cada página pode ter atributos de proteção independentes

Evolução da Paginação no x86

- **Paginação de 32 bits (80386)**: Esquema de dois níveis com diretórios e tabelas de páginas
- **Páginas de 4MB (Pentium)**: Extensão PSE (Page Size Extension) para suportar páginas maiores
- **PAE (Physical Address Extension)**: Estende o endereçamento físico para 36 bits (64GB) mesmo em modo 32 bits
- **Paginação em Modo Longo**: Esquema de quatro níveis (IA-32e) ou cinco níveis (mais recente), suportando endereçamento de até 57 bits

TLB (Translation Lookaside Buffer)

O TLB é uma memória cache especializada que armazena traduções recentes de endereços virtuais para físicos:

- **Objetivo**: Acelerar o processo de tradução de endereços evitando múltiplos acessos à memória para consultar tabelas de páginas
- **Invalidação**: Quando tabelas de página são modificadas, entradas correspondentes no TLB devem ser invalidadas através de instruções específicas (como INVLPG) ou reinicialização completa do TLB
- **Tipos**: Processadores modernos geralmente têm TLBs separados para instruções (ITLB) e dados (DTLB), além de múltiplos níveis de TLB

1.2.4 Registradores de Controle

Os registradores de controle são fundamentais para o gerenciamento de memória no x86:

- **CR0**: Controla modos de operação do processador
  - Bit 0 (PE): Habilita o Modo Protegido quando setado
  - Bit 31 (PG): Habilita a paginação quando setado

  - Bit 16 (WP): Quando setado, impede o kernel de escrever em páginas somente-leitura

- **CR2**: Contém o endereço linear que causou uma falha de página
- **CR3**: Contém o endereço físico da tabela de diretório de páginas (PDBR - Page Directory Base Register)
- **CR4**: Controla extensões da arquitetura
  - Bit 4 (PSE): Habilita suporte a páginas de 4MB em modo 32 bits
  - Bit 5 (PAE): Habilita Physical Address Extension
  - Bit 7 (PGE): Habilita suporte a páginas globais

1.2.5 Falhas de Página

As falhas de página são exceções (interrupção 14) geradas pelo processador quando ocorre um problema durante a tradução de endereços:

- **Causas comuns**:
  - Página não presente na memória (bit P = 0)
  - Violação de privilégios (tentativa de escrita em página somente-leitura)
  - Acesso em nível de privilégio inadequado
- **Informações disponíveis**:
  - CR2: Contém o endereço linear que causou a falha
  - Código de erro na pilha: Fornece detalhes sobre o tipo de falha
  - Bit 0: Indica se a página não estava presente (0) ou houve violação de proteção (1)
  - Bit 1: Indica se foi tentativa de escrita (1) ou leitura (0)
  - Bit 2: Indica se o acesso foi em modo usuário (1) ou supervisor (0)

1.2.6 Usos Avançados da Paginação

A paginação permite implementar diversas funcionalidades avançadas em sistemas operacionais:

- **Memória sob demanda**: Páginas são alocadas apenas quando necessárias
- **Copy-on-write**: Páginas são compartilhadas entre processos até que um deles tente escrever

- **Swapping**: Páginas pouco usadas podem ser movidas para armazenamento secundário
- **Isolamento de processos**: Cada processo tem seu próprio espaço de endereçamento virtual
- **Compartilhamento controlado**: Bibliotecas e recursos podem ser compartilhados entre processos
- **ASLR (Address Space Layout Randomization)**: Randomização da localização de áreas de memória para dificultar exploits

1.2.7 Considerações de Desempenho

O sistema de memória é frequentemente um gargalo de desempenho, e várias técnicas são usadas para otimizá-lo:

- **Hierarquia de cache**: L1, L2, L3 para reduzir latência de acesso
- **TLB**: Reduz o custo de tradução de endereços virtuais
- **Páginas grandes**: Reduzem o overhead do TLB ao cobrir mais memória com menos entradas
- **Prefetching**: Antecipação de acessos à memória para esconder latência
- **Alinhamento**: Dados alinhados são acessados mais eficientemente

O gerenciamento eficaz da memória é essencial para o desempenho do sistema como um todo, e compreender os mecanismos de baixo nível da arquitetura x86 permite implementações mais eficientes tanto em nível de sistema operacional quanto em aplicações de alta performance.

1.3 Sistemas Operacionais e Gerenciamento de Memória

Os sistemas operacionais modernos têm como um dos seus papéis fundamentais o gerenciamento eficiente da memória. Esta seção explora como os sistemas operacionais utilizam os mecanismos de hardware disponíveis na arquitetura x86 para implementar seus próprios sistemas de gerenciamento de memória.

1.3.1 Hierarquia de Memória

Os sistemas operacionais precisam lidar com uma hierarquia complexa de memória:

- **Registradores**: Gerenciados pelo compilador e não diretamente pelo SO
- **Cache L1, L2, L3**: Gerenciados pelo hardware, com alguma influência do SO
- **RAM**: Principal área de gerenciamento do SO
- **Memória de troca (swap)**: Extensão da RAM em armazenamento secundário
- **Sistemas de arquivos**: Armazenamento persistente mapeável em memória

1.3.2 Abstração de Memória

O sistema operacional proporciona diversas abstrações de memória para os aplicativos:

- **Espaço de endereçamento virtual**: Cada processo tem a ilusão de um espaço de endereçamento contíguo e privado
- **Isolamento entre processos**: Impede que processos acessem a memória uns dos outros sem autorização explícita
- **Memória compartilhada**: Mecanismos para compartilhamento controlado de memória entre processos
- **Alocação dinâmica**: APIs como `malloc()` e `free()` para gerenciamento de memória em tempo de execução

1.3.3 Layout da Memória do Processo

Em sistemas x86, o espaço de endereçamento de um processo tipicamente inclui:

- **Segmento de texto**: Código executável, geralmente somente leitura
- **Segmento de dados**: Variáveis globais e estáticas inicializadas
- **Segmento BSS**: Variáveis globais e estáticas não inicializadas

- **Heap**: Área para alocação dinâmica, gerenciada por `malloc()` e similares
- **Mapeamentos**: Bibliotecas compartilhadas, arquivos mapeados em memória
- **Stack**: Pilha para chamadas de função, variáveis locais e contexto

1.3.4 Estratégias de Gerenciamento

Os sistemas operacionais implementam diversas estratégias para gerenciar a memória:

- **Alocação sob demanda**: Páginas são realmente alocadas apenas quando acessadas pela primeira vez (lazy allocation)
- **Copy-on-write (COW)**: Otimização onde páginas são compartilhadas entre processos até que um deles tente modificá-las
- **Page replacement**: Algoritmos para decidir quais páginas mover para o swap quando a memória está cheia (LRU, Clock, etc.)
- **Compactação de memória**: Técnicas como KSM (Kernel Samepage Merging) para identificar e fundir páginas idênticas

1.3.5 Memória Virtual

A memória virtual é a abstração central nos sistemas operacionais modernos:

- **Endereçamento virtual**: Processos usam endereços virtuais, traduzidos para físicos pelo MMU
- **Swapping**: Páginas pouco utilizadas são movidas para armazenamento secundário
- **Overcommit**: Alocação de mais memória virtual do que a memória física disponível
- **Memória sob demanda**: Páginas são carregadas do disco apenas quando necessárias

1.3.6 Implementação de Paginação nos SOs

Os sistemas operacionais x86 modernos implementam sofisticados sistemas de paginação:

- **Tabelas de página multinível**: Linux, Windows e outros SOs usam a estrutura multinível suportada pelo hardware x86
- **Estruturas de rastreamento**: O kernel mantém estruturas auxiliares para rastrear o estado de cada página física

- **Page fault handler**: Rotina crítica do kernel que determina como responder a cada falha de página
- **Compartilhamento de páginas**: Mecanismos para mapear a mesma página física em múltiplos espaços de endereçamento

1.3.7 Page Fault Handler

O tratador de falhas de página é um componente crítico do SO, que pode responder de várias formas:

- **Demand paging**: Carrega uma página do disco que foi previamente swapped out
- **Copy-on-write**: Cria uma cópia privada de uma página compartilhada quando ocorre tentativa de escrita
- **Alocação lazy**: Aloca uma página física para um endereço virtual previamente reservado mas não alocado
- **Stack growth**: Expande automaticamente a pilha quando necessário
- **Erro de segmentação**: Quando o acesso é inválido, sinaliza SIGSEGV ao processo

1.3.8 Serviços de Memória para Aplicações

Sistemas operacionais fornecem APIs para gerenciamento de memória:

- **POSIX**:
  - `malloc()`, `free()`: Alocação dinâmica
  - `mmap()`, `munmap()`: Mapeamento de memória
  - `brk()`, `sbrk()`: Modificação do limite do heap
  - `mprotect()`: Alteração das permissões de páginas
- **Windows**:
  - `VirtualAlloc()`, `VirtualFree()`
  - `HeapAlloc()`, `HeapFree()`
  - `MapViewOfFile()`

1.3.9 Segurança de Memória

Os sistemas operacionais implementam diversas técnicas para melhorar a segurança da memória:

- **ASLR (Address Space Layout Randomization)**: Randomiza a posição das regiões de memória para dificultar ataques
- **DEP/NX (Data Execution Prevention/No-eXecute)**: Impede a execução de código em páginas de dados
- **Stack Canaries**: Detecção de overflow de buffer na pilha
- **KASLR (Kernel ASLR)**: Randomiza a posição do código do kernel na memória
- **SMAP/SMEP**: Proteções para prevenir que o kernel acesse ou execute código de modo usuário inadvertidamente

1.3.10 Considerações em Ambientes Baremetal

Em ambientes baremetal ou em kernels simples, o programador precisa implementar seu próprio sistema de gerenciamento de memória:

- **Inicialização do hardware**: Configurar GDT, IDT, e habilitar paginação manualmente
- **Alocação básica**: Implementar alocadores simples para gerenciar memória física
- **Mapeamento manual**: Configurar tabelas de página diretamente, sem as abstrações de um SO completo
- **Tratamento limitado de falhas**: Implementar tratadores básicos para falhas de página

Esta abordagem ”mínima”é valiosa para compreender os mecanismos de baixo nível que normalmente são abstraídos pelos sistemas operacionais completos, e constitui um excelente exercício educacional para entender como o hardware e software interagem no gerenciamento de memória.

1.4 Paginação na Arquitetura x86

A paginação é o mecanismo fundamental de gerenciamento de memória nos processadores modernos x86, permitindo a implementação de memória virtual, proteção entre processos e diversas outras funcionalidades essenciais para sistemas operacionais. Esta seção explora em detalhes os mecanismos de paginação na arquitetura Intel IA-32/x86-64, conforme documentado no Volume 3 do Intel® 64 and IA-32 Architectures Software Developer's Manual.

1.4.1 Fundamentos da Paginação

A paginação divide a memória em unidades fixas chamadas páginas e proporciona um mapeamento flexível entre endereços lineares (virtuais) e físicos:

- **Páginas**: Blocos de memória de tamanho fixo, tipicamente 4 KiB na arquitetura x86
- **Frames**: As unidades físicas correspondentes às páginas virtuais
- **MMU (Memory Management Unit)**: Hardware responsável pela tradução automática de endereços
- **Tabelas de Página**: Estruturas de dados que definem o mapeamento entre endereços virtuais e físicos

A paginação oferece diversas vantagens:

- Isolamento entre processos
- Compartilhamento controlado de memória
- Implementação eficiente de memória virtual
- Proteção granular (permissões por página)
- Uso eficiente da memória física fragmentada

1.4.2 Habilitando a Paginação

Para habilitar a paginação em um processador x86, é necessário:

1. Preparar as estruturas de tabelas de página na memória
2. Carregar o endereço físico da tabela de diretório de páginas no registro CR3
3. Habilitar a paginação setando o bit PG (bit 31) no registro CR0

4. Opcionalmente, configurar recursos avançados via CR4 (PSE, PAE, etc.)

Esta sequência deve ser cuidadosamente implementada, pois após habilitar a paginação, o processador imediatamente começa a traduzir todos os endereços. Se o mapeamento inicial não estiver corretamente configurado, o processador não conseguirá continuar a execução.

Código de Exemplo para Habilitar Paginação

Para habilitar paginação com páginas de 4MB no modo de 32 bits:

1. Declarar e inicializar a tabela de diretório de páginas:
    - Alinhar em limite de 4KB: `__attribute__((aligned(4096)))`
    - Inicializar pelo menos a primeira entrada para mapear os primeiros 4MB: `(1 << 7) | (1 << 1) | (1 << 0)`
    - O bit 7 (PSE) indica página de 4MB, bit 1 (R/W) permite escrita, bit 0 (P) marca como presente
2. Carregar o endereço da tabela em CR3: `set_cr3((uint32_t)&tabela_paginas);`
3. Habilitar páginas de 4MB em CR4: `set_cr4(get_cr4() | (1 << 4));`
4. Habilitar paginação em CR0: `set_cr0(get_cr0() | (1 << 31));`

1.4.3 Estruturas de Paginação de 32 bits

Na arquitetura IA-32, a paginação pode ser configurada em diferentes modos:

Paginação de 32 bits Padrão (Páginas de 4KB)

Este modo utiliza um esquema de dois níveis:

- **Diretório de Páginas (PD)**: Tabela de nível superior com 1024 entradas de 4 bytes
- **Tabelas de Página (PT)**: Tabelas de segundo nível, cada uma com 1024 entradas de 4 bytes

Um endereço linear de 32 bits é dividido em:

- Bits 31-22 (10 bits): Índice no Diretório de Páginas
- Bits 21-12 (10 bits): Índice na Tabela de Páginas
- Bits 11-0 (12 bits): Deslocamento dentro da página de 4KB

PSE (Page Size Extension) - Páginas de 4MB

Quando o bit PSE em CR4 está habilitado:

- O processador suporta páginas de 4MB
- O esquema é simplificado para um único nível
- Um endereço linear de 32 bits é dividido em:
  - Bits 31-22 (10 bits): Índice no Diretório de Páginas
  - Bits 21-0 (22 bits): Deslocamento dentro da página de 4MB

PAE (Physical Address Extension)

O PAE estende o endereçamento físico para 36 bits (permitindo acessar até 64GB de RAM):

- Esquema de três níveis
- Diretório de Diretórios de Página (PDPT) com 4 entradas de 8 bytes
- Diretórios de Páginas, cada um com 512 entradas de 8 bytes
- Tabelas de Página, cada uma com 512 entradas de 8 bytes
- Suporta páginas de 4KB e 2MB (com PSE)

1.4.4 Formato das Entradas de Tabelas de Página

Entrada de Diretório de Página (PDE) em Modo 32 bits

Uma entrada de 32 bits no diretório de páginas possui os seguintes campos:

- Bits 31-12: Endereço físico base da tabela de páginas (para páginas de 4KB) ou da página (para 4MB)
- Bit 7 (PS): Page Size - 0 para 4KB, 1 para 4MB
- Bit 6 (D): Dirty - Indica se a página foi escrita
- Bit 5 (A): Accessed - Indica se a página foi acessada
- Bit 4 (PCD): Page Cache Disable - Desabilita cache para esta página
- Bit 3 (PWT): Page Write Through - Configura cache para write-through
- Bit 2 (U/S): User/Supervisor - 0 para página de supervisor, 1 para página de usuário
- Bit 1 (R/W): Read/Write - 0 para somente leitura, 1 para leitura/escrita
- Bit 0 (P): Present - 1 indica página presente na memória física

Entrada de Tabela de Página (PTE) em Modo 32 bits

Similar à PDE, mas sempre referencia uma página de 4KB:

- Bits 31-12: Endereço físico base da página de 4KB
- Os bits de controle (A, D, PCD, PWT, U/S, R/W, P) têm o mesmo significado que na PDE

1.4.5 Tradução de Endereços

O processo de tradução de endereços usando paginação de 32 bits acontece da seguinte forma:

1. O processador extrai os bits 31-22 do endereço linear para obter o índice no diretório de páginas
2. Multiplica este índice por 4 (tamanho de cada entrada) e soma ao endereço base do diretório (armazenado em CR3)
3. Lê a entrada do diretório (PDE) neste endereço
4. Verifica se a PDE está presente (bit P) e se tem as permissões apropriadas
5. Se a PDE indica uma página de 4MB (bit PS=1):
    - O endereço físico é formado combinando os bits 31-22 da PDE com os bits 21-0 do endereço linear
6. Se a PDE indica uma tabela de páginas (bit PS=0):
    - Extrai os bits 21-12 do endereço linear para obter o índice na tabela de páginas
    - Multiplica este índice por 4 e soma ao endereço base da tabela (obtido da PDE)
    - Lê a entrada da tabela (PTE) neste endereço
    - Verifica se a PTE está presente e tem as permissões apropriadas
    - O endereço físico é formado combinando os bits 31-12 da PTE com os bits 11-0 do endereço linear

1.4.6 Falhas de Página

Uma falha de página (Page Fault) ocorre quando há um problema durante a tradução de endereço:

- O processador gera uma exceção de falha de página (interrupção 14)
- O código de erro empilhado fornece informações sobre a causa:
  - Bit 0: 0 se a página não estava presente, 1 se foi violação de proteção
  - Bit 1: 0 se foi acesso de leitura, 1 se foi acesso de escrita
  - Bit 2: 0 se foi acesso em modo kernel, 1 se foi acesso em modo usuário
  - Bit 3: 1 se a falha envolveu bits reservados
  - Bit 4: 1 se a falha ocorreu durante uma busca de instrução
- O endereço que causou a falha é armazenado no registro CR2

O tratador de falhas de página pode tomar diferentes ações:

- Carregar a página da memória de troca (implementando memória virtual)
- Alocar uma nova página para alocação sob demanda
- Criar uma cópia da página para implementar copy-on-write
- Terminar o processo se o acesso for inválido (segmentation fault)

1.4.7 Translation Lookaside Buffer (TLB)

O TLB é uma cache de traduções de endereços que acelera o processo de paginação:

- Armazena as traduções mais recentes de endereços lineares para físicos
- É consultado antes das tabelas de página na memória
- Se uma tradução está presente no TLB (hit), a consulta às tabelas é evitada
- É gerenciado automaticamente pelo hardware, mas requer intervenção do software em alguns casos

Quando as tabelas de página são modificadas, o TLB deve ser invalidado:

- **Invalidação específica**: A instrução INVLPG invalida uma entrada específica do TLB
- **Invalidação global**: Recarregar CR3 invalida todo o TLB (exceto entradas marcadas como globais)

1.4.8 Flags e Bits de Controle

Além dos bits nas entradas das tabelas, vários bits nos registradores de controle afetam o comportamento da paginação:

- **CR0.WP** (Write Protect, bit 16):
  - Quando 0, o kernel pode escrever em páginas somente-leitura
  - Quando 1, as proteções de escrita são aplicadas mesmo em modo kernel
- **CR4.PSE** (Page Size Extensions, bit 4):
  - Quando 1, habilita suporte a páginas de 4MB em modo 32 bits
- **CR4.PGE** (Page Global Enable, bit 7):
  - Quando 1, permite marcar páginas como globais (não são invalidadas no TLB durante trocas de CR3)
- **CR4.PAE** (Physical Address Extension, bit 5):
  - Quando 1, habilita PAE para suportar endereçamento físico estendido

1.4.9 Técnicas Avançadas

Mapeamentos Múltiplos

É possível mapear o mesmo frame físico em múltiplos endereços virtuais:

- Útil para compartilhamento de memória entre processos
- Permite ter diferentes permissões para o mesmo conteúdo físico
- Facilita operações copy-on-write

Páginas de Kernel Mapeadas em Todo Espaço de Endereçamento

Sistemas operacionais modernos normalmente:

- Dividem o espaço de endereço virtual em porções de usuário e kernel
- Mapeiam o código e dados do kernel em todos os espaços de endereço
- Marcam essas páginas como globais para evitar invalidações do TLB durante trocas de contexto

Paginação Aninhada e Virtualização

Em ambientes virtualizados:

- O hardware moderno suporta paginação aninhada (NPT/EPT)
- Permite que máquinas virtuais usem paginação sem overhead excessivo
- Adiciona uma camada extra de tradução: GVA → GPA → HPA (Guest Virtual → Guest Physical → Host Physical)

1.4.10 Considerações de Implementação Prática

Ao implementar paginação em um ambiente baremetal ou kernel simples:

- **Identity mapping inicial**: Mapeie os primeiros MB da memória física para endereços virtuais idênticos
- **Mapeamento do código**: Garanta que o código do kernel permaneça acessível após habilitar paginação
- **Alinhamento**: Todas as estruturas de paginação devem estar alinhadas em limites de página (4KB)
- **Tratamento de interrupções**: Configure o tratador de falhas de página antes de experimentar com mapeamentos
- **Inicialização cuidadosa**: A sequência de inicialização deve garantir que a próxima instrução após habilitar paginação seja acessível

A paginação é uma das funcionalidades mais poderosas e complexas da arquitetura x86. Sua compreensão detalhada é fundamental para o desenvolvimento de sistemas operacionais e para a implementação de funcionalidades avançadas de gerenciamento de memória.

Capítulo 2

Referências