Gerenciamento de Memória em Linux e Windows

Introdução **1**
Conceitos Gerais e Abordagens 1
Linux: Gerenciamento de Memória 2
Windows: Gerenciamento de Memória 3
Comparativo e Conclusão **4**
Referências **5**

Introdução

Os sistemas operacionais Linux e Windows são amplamente utilizados em uma vasta gama de dispositivos, desde servidores de alta performance e desktops até sistemas embarcados. Ambos implementam mecanismos sofisticados de gerenciamento de memória para fornecer aos processos um ambiente de execução estável, eficiente e seguro. Embora os conceitos fundamentais de gerenciamento de memória, como paginação e memória virtual, sejam comuns, as implementações específicas, as estruturas de dados internas, as chamadas de sistema (APIs) e as políticas de gerenciamento podem variar significativamente entre eles. Este capítulo explora como o gerenciamento de memória é realizado especificamente nos sistemas operacionais Linux e Windows, detalhando seus conceitos, as chamadas de sistema disponíveis para os desenvolvedores e aspectos de suas implementações internas. Compreender essas especificidades é crucial para desenvolvedores de sistemas e aplicações que buscam otimizar o uso de memória e o desempenho em cada plataforma.

Conceitos Gerais e Abordagens

Tanto Linux quanto Windows utilizam memória virtual paginada como o principal mecanismo de gerenciamento de memória. Isso permite que cada processo tenha seu próprio espaço de endereçamento virtual privado, que é mapeado para a memória física (RAM) sob demanda. Ambos os sistemas também gerenciam a memória física dividindo-a em quadros (frames) e utilizam algoritmos de substituição de páginas quando a memória física se torna escassa.

Linux: Gerenciamento de Memória

O gerenciamento de memória no Linux é complexo e evoluiu consideravelmente ao longo das versões do kernel. Ele é projetado para ser eficiente e escalável em uma ampla variedade de arquiteturas de hardware.

Principais Conceitos e Componentes:

- **Zonas de Memória (Memory Zones):** O Linux divide a memória física em zonas (e.g., ZONE_DMA, ZONE_NORMAL, ZONE_HIGHMEM em arquiteturas de 32 bits) para lidar com as particularidades de diferentes regiões da memória física e suas capacidades de endereçamento por dispositivos de hardware.
- **Buddy System Allocator:** Usado para alocar e desalocar blocos de páginas fisicamente contíguas. Ele gerencia blocos de memória de tamanhos que são potências de 2 (e.g., 1, 2, 4, 8, 16 páginas).
- **Slab Allocator (e suas variantes como SLUB, SLOB):** Usado para alocar memória para objetos do kernel de tamanhos específicos e frequentemente usados (e.g., estruturas de dados do kernel como task_struct, inode). Ele visa reduzir a fragmentação interna e melhorar a eficiência da alocação/desalocação, mantendo caches de objetos pré-inicializados.
- **Tabelas de Páginas:** O Linux usa uma estrutura de tabela de páginas multinível (geralmente 3 ou 4 níveis em arquiteturas de 32 bits, e 4 ou 5 níveis em arquiteturas de 64 bits) para mapear endereços virtuais para endereços físicos. Cada processo tem seu próprio conjunto de tabelas de páginas.
- **Page Cache:** O Linux mantém um cache de páginas de disco na memória RAM para acelerar o acesso a arquivos e dados de dispositivos de bloco. As operações de leitura e escrita em arquivos geralmente passam pelo page cache.
- **Swapping:** Quando a memória física está cheia, páginas inativas podem ser movidas para uma partição de swap ou arquivo de swap no disco. O Linux usa uma variação do algoritmo LRU (Least Recently Used), como o algoritmo de duas listas (ativo/inativo), para selecionar páginas candidatas à substituição.
- **Out-of-Memory (OOM) Killer:** Se o sistema ficar criticamente sem memória e não puder liberar mais páginas através do swapping, o OOM killer é invocado para selecionar e matar um ou mais processos para liberar memória e evitar um travamento completo do sistema.
- **Copy-on-Write (COW):** Usado durante a chamada fork(). Em vez de duplicar todas as páginas do processo pai para o filho imediatamente, as páginas são compartilhadas. Uma cópia real da página só é feita se o pai ou o filho tentarem modificar a página compartilhada.
- **Huge Pages (Páginas Enormes):** Suporte para páginas de tamanhos maiores (e.g., 2MB, 1GB) para reduzir a sobrecarga do TLB e das tabelas de páginas para aplicações que manipulam grandes volumes de memória.

Chamadas de Sistema (APIs) Relevantes no Linux:

- brk() e sbrk(): Funções mais antigas para aumentar ou diminuir o tamanho do segmento de dados (heap) do processo. sbrk(0) retorna o final atual do heap.
- mmap(): Mapeia arquivos ou dispositivos na memória. Pode ser usado para criar regiões de memória compartilhada entre processos ou para alocar memória anônima (não associada a um arquivo). É a principal forma moderna de alocação de memória no Linux.
- munmap(): Desmapeia uma região de memória previamente mapeada por mmap().

- mprotect(): Altera as permissões de acesso (leitura, escrita, execução) de uma região de memória mapeada.
- mlock() e munlock(): Bloqueiam ou desbloqueiam páginas na memória física, impedindo que sejam paginadas para o disco. Útil para aplicações de tempo real.
- /proc/[pid]/maps e /proc/[pid]/smaps: Arquivos virtuais no sistema de arquivos /proc que fornecem informações detalhadas sobre o mapa de memória de um processo específico.

Windows: Gerenciamento de Memória

O gerenciador de memória do Windows é responsável por fornecer serviços de memória virtual, gerenciar a memória física e o pagefile.

Principais Conceitos e Componentes:

- **Virtual Address Descriptors (VADs):** O gerenciador de memória do Windows usa uma árvore de VADs auto-balanceada para cada processo, a fim de rastrear as regiões de endereços virtuais que foram reservadas ou alocadas.
- **Page Frame Number (PFN) Database:** Uma estrutura de dados central que descreve o estado de cada página física na memória (e.g., livre, em uso, modificada, em transição).
- **Working Set:** O conjunto de páginas físicas atualmente residentes na memória para um processo específico. O Windows tenta manter o working set de cada processo na memória para minimizar as faltas de página. Existe um working set mínimo e máximo para cada processo.
- **Paging:** O Windows usa um algoritmo de substituição de páginas que é uma variação do LRU, baseado em um esquema de envelhecimento e no estado das páginas (ativas/inativas). As páginas são movidas para listas de espera (standby, modified, modified no-write) antes de serem reutilizadas ou escritas no pagefile.
- **Pagefile (Arquivo de Paginação):** Onde as páginas modificadas que são removidas da memória física são armazenadas. O Windows pode suportar múltiplos pagefiles.
- **Memory Sections (Seções de Memória):** Objetos do kernel que representam blocos de memória que podem ser compartilhados entre processos ou mapeados para arquivos (memory-mapped files). A alocação de memória virtual no Windows geralmente envolve a criação e o mapeamento de seções.
- **Heaps:** O Windows fornece gerenciamento de heap tanto no nível do sistema (para o kernel) quanto para processos de usuário. Cada processo tem um heap padrão, e pode criar heaps privados adicionais.
- **Address Windowing Extensions (AWE):** Permite que aplicações de 32 bits acessem mais de 4GB de memória física em sistemas que a suportam, mapeando

Chamadas de Sistema (APIs) Relevantes no Windows (Win32 API):

- VirtualAlloc(): Reserva ou comita uma região de páginas no espaço de endereçamento virtual do processo.

- VirtualFree(): Libera ou descomita uma região de páginas.
- VirtualProtect(): Altera a proteção de acesso de uma região de páginas comitadas.
- VirtualQuery(): Retorna informações sobre uma região de páginas no espaço de endereçamento virtual do processo.
- CreateFileMapping() e MapViewOfFile(): Usadas para criar e mapear seções de memória (memory-mapped files), que podem ser usadas para memória compartilhada ou para mapear arquivos em memória.
- UnmapViewOfFile(): Desmapeia uma view de um arquivo mapeado.
- HeapCreate(), HeapAlloc(), HeapFree(), HeapDestroy(): Funções para criar e gerenciar heaps privados no espaço do usuário.

Comparativo e Conclusão

Tanto Linux quanto Windows implementam sistemas de gerenciamento de memória virtual robustos e complexos, baseados em paginação sob demanda. Ambos utilizam mecanismos como tabelas de páginas, TLBs, algoritmos de substituição de páginas e swapping/paging para gerenciar eficientemente a memória física e fornecer um grande espaço de endereçamento virtual para os processos.

As principais diferenças residem nas estruturas de dados internas específicas (e.g., zonas de memória e buddy/slab no Linux vs. VADs e PFN database no Windows), nas políticas de gerenciamento (e.g., algoritmos de substituição de páginas, gerenciamento do working set) e nas APIs expostas aos desenvolvedores (mmap e família no Linux vs. VirtualAlloc e família no Windows).

O Linux tende a ser mais transparente em suas implementações internas, com muito do seu código fonte disponível e extensivamente documentado pela comunidade. O Windows, embora com partes de sua arquitetura bem documentadas (especialmente através de livros como “Windows Internals”), mantém muitos detalhes de implementação proprietários.

Ambos os sistemas evoluíram para suportar arquiteturas modernas de 64 bits, grandes quantidades de RAM e funcionalidades avançadas como páginas enormes/grandes para otimizar o desempenho de aplicações intensivas em memória. A compreensão das particularidades do gerenciamento de memória em cada sistema operacional é fundamental para o desenvolvimento de software eficiente e para a administração e tuning de sistemas.

Referências

- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.
- Tanenbaum, A. S., & Bos, H. (2015). *Modern Operating Systems* (4th ed.). Pearson Education.
- Love, R. (2010). *Linux Kernel Development* (3rd ed.). Addison-Wesley Professional.

- Russinovich, M. E., Solomon, D. A., & Ionescu, A. (2012). *Windows Internals, Part 1* (6th ed.). Microsoft Press.
- Bovet, D. P., & Cesati, M. (2005). *Understanding the Linux Kernel* (3rd ed.). O'Reilly Media.