RESUMO DO LIVRO FUNDAMENTOS DE SISTEMAS OPERACIONAIS 9ª

A gestão de memória é essencialmente importante para uma operação eficiente de um sistema de computação moderno, e envolve várias abordagens, como a paginação e a segmentação. Cada método possui suas vantagens e desvantagens, e a escolha do método adequado depende, principalmente, do design do hardware do sistema. A integração entre o hardware e o gerenciamento de memória do sistema operacional é frequentemente necessária para suportar esses métodos.

A memória, constituída por um grande array de bytes, é fundamental para a operação do sistema, sendo o local onde a CPU busca instruções e dados para executar programas. A CPU acessa diretamente a memória principal e os registradores embutidos, mas não pode acessar diretamente os endereços de disco. Para garantir um acesso eficiente, a memória principal deve estar disponível, e qualquer dado não presente nela deve ser transferido antes de ser utilizado pela CPU. Registradores da CPU são acessados rapidamente, normalmente dentro de um ciclo de relógio da CPU, enquanto o acesso à memória principal pode requerer vários ciclos, o que pode causar interrupções na execução da CPU. Para mitigar isso, uma memória rápida é colocada entre a CPU e a memória principal. Além da velocidade de acesso à memória, é crucial assegurar a proteção correta do sistema. Cada processo deve ter um espaço de memória separado para proteger o sistema operacional de acessos indevidos por processos de usuário e para evitar que processos de usuário interfiram uns nos outros. Isso é realizado através de registradores base e limite, que definem o intervalo de endereços legais que um processo pode acessar. Por exemplo, se o registrador base contiver o endereço 300040 e o registrador limite for 120900, o programa poderá acessar legalmente os endereços de 300040 a 420939.

O processo de vinculação de endereços é fundamental para a execução de programas. Um programa geralmente reside no disco como um arquivo binário executável e deve ser carregado na memória para ser executado. A vinculação de endereços pode ocorrer em diferentes momentos: tempo de compilação, tempo de carga ou tempo de execução. No tempo de compilação, se o local na memória onde o processo será carregado é conhecido, um código absoluto pode ser gerado. Se não for conhecido, a vinculação é feita no tempo de carga, gerando código realocável. Se o processo pode ser movido durante sua execução, a vinculação é feita em tempo de execução, exigindo suporte de hardware específico.

Um endereço gerado pela CPU é denominado endereço lógico, enquanto o endereço visto pela unidade de memória é o endereço físico. Nos métodos de vinculação de tempo de compilação e tempo de carga, os endereços lógicos e físicos são idênticos. Porém, na vinculação em tempo de execução, os endereços lógicos e físicos são diferentes. Esse mapeamento é realizado por uma unidade de gerenciamento da memória (MMU), que converte endereços lógicos em físicos dinamicamente. Por exemplo, se a base for 14000, uma tentativa de acessar a locação 0 será realocada para 14000, e um acesso à locação 346 será mapeado para 14346.

Para uma utilização mais eficiente da memória, pode-se usar a carga dinâmica, onde uma rotina só é carregada quando é chamada. Isso é útil para grandes volumes de código necessários em situações raras, como rotinas de erro. A carga dinâmica não requer suporte

especial do sistema operacional, mas o sistema pode fornecer rotinas de biblioteca para facilitar sua implementação. As bibliotecas vinculadas dinamicamente são incorporadas aos programas durante a execução, economizando espaço em disco e memória, pois múltiplos programas podem compartilhar a mesma biblioteca. Quando uma rotina de biblioteca é necessária, um stub verifica se já está na memória e, se não estiver, carrega a rotina. Após isso, a rotina é executada diretamente, sem o custo adicional da vinculação dinâmica.

Esses conceitos são essenciais para o gerenciamento eficiente da memória em sistemas operacionais, permitindo a execução segura e eficaz de múltiplos processos, otimizando a utilização da CPU e garantindo a proteção e integridade do sistema.

Inicialmente, os algoritmos de gerenciamento de memória discutidos anteriormente são necessários porque as instruções devem estar na memória física para serem executadas. Tradicionalmente, isso requer que todo o espaço de endereçamento lógico de um programa esteja presente na memória física, o que pode ser limitador e não eficiente.

Programas reais muitas vezes não necessitam de estar totalmente carregados na memória física. Por exemplo, códigos para lidar com erros raramente são executados, arrays e tabelas geralmente são superdimensionados, e algumas funcionalidades de programas são raramente utilizadas. Além disso, mesmo quando um programa inteiro é necessário, ele não precisa estar todo na memória ao mesmo tempo. A capacidade de executar um programa que não esteja completamente na memória física traz inúmeros benefícios como, flexibilidade na programação, em que programadores podem escrever programas para espaços de endereçamento muito grandes, sem se preocupar com a quantidade de memória física disponível, maior eficiência de recursos, mais programas podem ser executados simultaneamente, redução de operações de I/O, ou seja, menos operações são necessárias para carregar ou permutar programas na memória.

O espaço de endereçamento virtual é a visão lógica de como um processo é armazenado na memória, começando de um endereço lógico específico e ocupando uma memória contígua. No entanto, fisicamente, a memória pode estar organizada em quadros de páginas, que não são contíguos. A responsabilidade de mapear essas páginas lógicas para quadros físicos é da unidade de gerenciamento de memória (MMU).

Além disso, a memória virtual permite o compartilhamento de arquivos e memória entre processos através do compartilhamento de páginas. Bibliotecas do sistema, por exemplo, podem ser mapeadas para o espaço de endereçamento virtual de vários processos, permitindo que as páginas físicas sejam compartilhadas. Processos também podem compartilhar regiões de memória para comunicação eficiente, acelerando a criação de novos processos. Esses benefícios e outros serão explorados em detalhes ao longo do capítulo, especialmente focando na implementação da memória virtual por meio da paginação por demanda.

A paginação por demanda é uma técnica utilizada em sistemas de memória virtual, onde páginas de um programa são carregadas na memória física somente quando necessário, em vez de carregar todo o programa de uma vez. Isso permite uma utilização mais eficiente da memória, evitando a carga de partes do programa que podem nunca ser usadas. Quando um processo é iniciado, o paginador decide quais páginas são necessárias e as traz para a memória, evitando carregar páginas desnecessárias. Um esquema de bit válido-inválido é usado para diferenciar páginas na memória e no disco. Se um processo tentar acessar uma página que não está na memória, ocorre um erro de página, e o sistema operacional deve tratar esse erro, trazendo a página necessária para a memória.

A paginação por demanda pode afetar significativamente o desempenho do sistema. O tempo de acesso efetivo da memória é calculado considerando a probabilidade de ocorrência de erros de página (p). Um aumento na taxa de erros de página pode levar a uma degradação significativa do desempenho. Por exemplo, se um em cada 1.000 acessos à memória resultar em um erro de página, o tempo de acesso efetivo pode aumentar dramaticamente. Assim, a eficiência do sistema depende da minimização da taxa de erros de página e da otimização do tempo de serviço.

Os sistemas podem melhorar a eficiência da paginação por demanda utilizando o espaço de permuta para armazenar páginas de processos. A memória secundária (disco) geralmente é usada para manter páginas não presentes na memória principal. A paginação por demanda pode ser otimizada copiando uma imagem do arquivo inteiro para o espaço de permuta na inicialização do processo e executando a paginação a partir desse espaço.

Alguns sistemas limitam o uso do espaço de permuta para reduzir o overhead. Sistemas operacionais móveis, por exemplo, geralmente não suportam a permuta, executando a paginação por demanda diretamente a partir do sistema de arquivos e liberando páginas somente de leitura quando a memória é limitada. Portanto, a paginação por demanda é uma técnica poderosa para gerenciar a memória de forma eficiente, carregando apenas as páginas necessárias para a execução de um processo. No entanto, a taxa de erros de página e o tempo de serviço desses erros devem ser cuidadosamente gerenciados para manter um bom desempenho do sistema.

Na Seção 9.2, abordamos a criação rápida de processos através da técnica de cópia-após-gravação, uma estratégia que minimiza o número de novas páginas alocadas para um processo recém-criado. Tradicionalmente, a chamada de sistema `fork()` criava uma cópia do espaço de endereçamento do processo-pai para o processo-filho, o que podia resultar em overhead se o processo-filho invoca se imediatamente `exec()` para substituir seu espaço de endereçamento. ou seja, a técnica de cópia-após-gravação permite que o processo-pai e o processo-filho compartilhem inicialmente as mesmas páginas, marcadas como páginas de cópia-após-gravação.

Quando um processo altera uma página compartilhada, uma cópia dessa página é criada, permitindo que os processos mantenham seus espaços de endereçamento separados somente quando necessário. Por exemplo, ao modificar uma página marcada como cópia-após-gravação, o sistema operacional cria uma cópia da página para o processo-filho, garantindo que as modificações não afetem o processo-pai. Isso é especialmente útil quando um processo-filho tenta modificar partes da pilha, pois apenas as páginas modificadas são copiadas, enquanto as não modificadas permanecem compartilhadas.

Uma das soluções encontradas para os erros de paginação por demanda foi a substituição de páginas, que por sua vez, aborda a taxa de erros de página que ocorre quando as páginas de memória que um processo precisa não estão na memória principal, resultando em falhas de página. Na prática, os processos frequentemente não utilizam todas as suas páginas alocadas, e a paginação por demanda permite economizar recursos ao carregar apenas as páginas necessárias. Esse mecanismo melhora a utilização da CPU e o throughput do sistema.

Quando se aumenta a multiprogramação, a memória pode ser super alocada, resultando na possibilidade de não haver quadros suficientes disponíveis quando todos os processos tentam usar suas páginas ao mesmo tempo. Além disso, a memória do sistema é usada também para buffers de I/O, aumentando a complexidade da alocação de memória.

Quando ocorre um erro de página e não há quadros livres, o sistema operacional deve decidir qual página existente na memória será substituída. Isso envolve:

1. Encontrar a página desejada no disco.

2. Procurar um quadro livre.

3. Se não houver quadro livre, selecione uma "página vítima" usando um algoritmo de substituição de páginas.

4. Gravar a página vítima no disco, se necessário.

5. Transferir a página desejada para o quadro liberado.

6. Atualizar as tabelas de páginas e quadros.

7. Retomar o processo do usuário.

Vários algoritmos de substituição de páginas foram desenvolvidos para otimizar o uso da memória:

- FIFO (First-In-First-Out): Substitui a página mais antiga na memória. Embora simples, pode resultar na substituição de páginas que ainda são frequentemente usadas, aumentando a taxa de erros de página.

- Substituição de Páginas Ótima (OPT ou MIN): O algoritmo de substituição de páginas ótimo, também conhecido como OPT ou MIN, substitui a página que não será utilizada pelo maior período de tempo no futuro. Este algoritmo garante a menor taxa de erros de página possível para um número fixo de quadros e não sofre da anomalia de Belady.

- Substituição de Páginas LRU (Least-Recently-Used): O algoritmo LRU é uma aproximação prática do algoritmo ótimo, substituindo a página que não foi utilizada pelo maior período de tempo recente. Associando cada página à última vez em que foi usada, o LRU seleciona a página menos recentemente utilizada para substituição. Este algoritmo também não sofre da anomalia de Belady e é amplamente considerado eficiente, sendo utilizado frequentemente em sistemas operacionais.

- Substituição de Páginas por Aproximação ao LRU: Dado que poucos sistemas fornecem suporte de hardware suficiente para uma verdadeira implementação do LRU, várias aproximações são usadas. Uma dessas aproximações é o algoritmo dos bits de referência adicionais, onde um byte de 8 bits é usado para registrar os bits de referência em intervalos regulares. Este byte contém um histórico de uso da página, permitindo identificar a página LRU para substituição.

- Outra aproximação é o algoritmo da segunda chance. Este é um algoritmo FIFO modificado onde, ao selecionar uma página para substituição, verifica-se seu bit de referência. Se o bit estiver ligado, a página recebe uma segunda chance e o bit é zerado. O algoritmo avança para a próxima página, continuando o processo até encontrar uma página com o bit de referência desligado. Uma forma comum de implementar este algoritmo é como uma fila circular com um ponteiro que avança zerando os bits de referência.

- Algoritmo da Segunda Chance Aperfeiçoado: Este algoritmo considera tanto o bit de referência quanto o bit de modificação, classificando as páginas em quatro categorias:

1. Nem recentemente utilizada nem modificada.

2. Não recentemente utilizada, mas modificada.

3. Recentemente utilizada, mas não modificada.

4. Recentemente utilizada e modificada.

A substituição de páginas segue o mesmo esquema do algoritmo da segunda chance, mas priorizando a substituição de páginas nas classes mais baixas, começando por páginas que não foram recentemente utilizadas nem modificadas, para minimizar operações de I/O.

Desse modo, a substituição de páginas é essencial para a paginação por demanda, permitindo que a memória lógica dos programas exceda a memória física disponível. A eficiência dos algoritmos de substituição impacta diretamente o desempenho do sistema, tornando a escolha e o desenvolvimento desses algoritmos uma tarefa crítica para os sistemas operacionais. Ademais, cada um desses algoritmos e suas aproximações buscam balancear eficiência e viabilidade prática na gestão de memória de sistemas operacionais. O algoritmo ótimo serve como padrão teórico, enquanto o LRU e suas variantes oferecem soluções pragmáticas que podem ser implementadas com o suporte adequado de hardware.

Na seção 9.5 vimos a alocação de quadros, que envolve decidir como distribuir a memória física disponível entre os vários processos. Em um sistema monousuário, a memória é dividida entre o sistema operacional e o processo do usuário. Por exemplo, se um sistema tem 128 KB de memória com páginas de 1KB, e o sistema operacional usa 35 KB, então 93 quadros ficam disponíveis para o processo do usuário. Inicialmente, todos os quadros livres são atribuídos ao processo, mas quando a lista de quadros livres se esgota, um algoritmo de substituição é usado para liberar quadros. Há variações dessa estratégia, como alocar o espaço de buffers e tabelas do sistema operacional a partir da lista de quadros livres, e manter um número mínimo de quadros livres para otimizar a resposta aos erros de página.

O número mínimo de quadros que um processo deve ter é determinado pela arquitetura do computador e pelo desempenho necessário. Por exemplo, se uma instrução referenciar um endereço na memória, precisamos de pelo menos dois quadros: um para a instrução e outro para a referência. Em casos mais complexos, como endereçamento indireto de vários níveis, o número de quadros necessários pode aumentar significativamente. Em arquiteturas como o PDP-11 ou o IBM 370, uma única instrução pode exigir até seis ou oito quadros. Para evitar problemas de desempenho, deve-se garantir que cada processo tenha um número mínimo adequado de quadros, de acordo com a arquitetura.

Ademais, existem várias maneiras de alocar quadros entre processos. A alocação igual divide os quadros igualmente entre os processos. Por exemplo, se há 93 quadros e cinco processos, cada um receberá 18 quadros, com três quadros restantes como buffer. No entanto, essa abordagem pode ser ineficiente se os processos tiverem diferentes necessidades de memória. A alocação proporcional, por outro lado, distribui a memória disponível de acordo com o tamanho de cada processo. Por exemplo, se um processo usa 10 KB e outro 127 KB em um sistema com 62 quadros, eles receberiam 4 e 57 quadros, respectivamente. Esse método garante que a memória seja utilizada de acordo com as necessidades dos processos.

Os algoritmos de substituição de páginas podem ser classificados em substituição global e local. Na substituição global, um processo pode substituir quadros de qualquer outro processo, permitindo maior flexibilidade e throughput do sistema. No entanto, isso pode causar variabilidade no desempenho dos processos, pois a taxa de erros de página de um processo pode ser afetada pelo comportamento de outros processos. A substituição local, por outro lado, limita a substituição de páginas ao conjunto de quadros alocados a cada processo, garantindo que a taxa de erros de página dependa apenas do comportamento do próprio processo. Embora a substituição local ofereça previsibilidade, a substituição global é geralmente preferida por seu melhor desempenho geral do sistema.

Em sistemas de acesso não uniforme à memória(NUMA), o tempo de acesso à memória varia dependendo da localização da memória e das CPUs. Em tais sistemas, as CPUs podem acessar a memória local mais rapidamente do que a memória em outras placas do sistema. Para otimizar o desempenho, os algoritmos de alocação de memória e escalonamento devem considerar essas variações de latência. A alocação de memória deve tentar manter os quadros o mais próximo possível da CPU que executa o processo. Sistemas como o Solaris utilizam grupos de latência (lgroups) para agrupar CPUs e memória próximas, minimizando a latência e maximizando a eficiência dos acessos ao cache. Isso é especialmente importante em sistemas com múltiplos threads, onde a distribuição equilibrada da carga de trabalho e da alocação de memória é crucial para o desempenho.

Quando um processo é alocado com menos quadros do que o necessário pela arquitetura do computador, a execução desse processo pode ser suspensa e suas páginas removidas da memória, liberando quadros alocados. Esse cenário leva a uma intensa atividade de paginação, chamada de atividade improdutiva(thrashing). Durante a atividade improdutiva, um processo gasta mais tempo paginando do que executando suas tarefas.

A atividade improdutiva é prejudicial ao desempenho do sistema. O sistema operacional monitora a utilização da CPU e, ao perceber uma utilização baixa, aumenta a multiprogramação introduzindo novos processos. Com um algoritmo global de substituição de páginas, as páginas são substituídas sem considerar o processo ao qual pertencem. Se um processo precisa de mais quadros, ele começa a falhar e a tomar quadros de outros processos, que também passam a falhar, gerando uma cadeia de falhas e intensa atividade de paginação.

Assim, para combater a atividade improdutiva, o modelo do conjunto de trabalho é utilizado. Esse modelo define que um processo deve ter quadros suficientes para acomodar sua localidade corrente — o conjunto de páginas ativamente usadas em um dado momento. Usando um parâmetro Δ, o conjunto de trabalho considera as Δ referências de página mais recentes para determinar quais páginas estão ativamente sendo usadas.

O tamanho do conjunto de trabalho de cada processo é crucial. Se a soma dos tamanhos dos conjuntos de trabalho de todos os processos ultrapassa o número total de quadros disponíveis, ocorre a atividade improdutiva. O sistema operacional monitora esses conjuntos de trabalho, ajustando a alocação de quadros conforme necessário e suspendendo processos quando a demanda por quadros excede a capacidade disponível.

A atividade improdutiva pode ser evitada garantindo que os processos tenham quadros suficientes para suas localidades correntes, utilizando o modelo do conjunto de trabalho. Isso otimiza a utilização da CPU e mantém o sistema funcionando eficientemente, evitando o estado onde processos passam mais tempo paginando do que executando tarefas úteis. A solução ideal é garantir memória física suficiente para evitar esses problemas sempre que possível. Tanto em dispositivos móveis quanto em sistemas de grande porte, ter memória adequada para manter todos os conjuntos de trabalho em memória simultaneamente proporciona a melhor experiência de usuário e otimiza o desempenho do sistema como um todo.

O mapeamento de arquivo para a memória é uma técnica poderosa para melhorar o desempenho do acesso a arquivos em sistemas operacionais. Em vez de usar chamadas de sistema como open(), read() e write() para acessar um arquivo no disco, podemos mapeá-lo para uma parte do espaço de endereçamento virtual da memória. Isso permite que operações de leitura e gravação no arquivo sejam tratadas como operações de acesso à memória, simplificando e acelerando o processo.

Quando um arquivo é mapeado para a memória, um bloco do disco é associado a uma página (ou páginas) na memória. O acesso inicial ao arquivo ainda pode envolver paginação por demanda, onde uma parte do arquivo é lida do sistema de arquivos para a memória. No entanto, subsequente leitura e escrita no arquivo são tratadas como operações de memória, evitando o overhead de chamadas de sistema separadas.

Um benefício significativo do mapeamento de arquivo para a memória é o compartilhamento eficiente de dados entre processos. Vários processos podem mapear o mesmo arquivo para a memória, permitindo que compartilhem dados através da memória compartilhada. Isso é útil para comunicação interprocessual e pode ser implementado usando chamadas de sistema específicas, como mmap() no Solaris ou APIs do Windows para memória compartilhada. Além disso, o I/O mapeado para a memória é uma técnica relacionada que reserva intervalos de endereços de memória para mapear diretamente registradores de dispositivos de I/O, como controladores de vídeo, portas seriais e paralelas. Isso simplifica as operações de I/O, especialmente em dispositivos com tempos de resposta rápidos.

Em resumo, o mapeamento de arquivo para a memória e o I/O mapeado para a memória são estratégias eficientes para melhorar o desempenho e simplificar o acesso a arquivos e dispositivos de I/O em sistemas computacionais. Essas técnicas são amplamente utilizadas em sistemas operacionais modernos para otimizar o gerenciamento de recursos e melhorar a eficiência das operações de entrada e saída.

O processo de alocação de memória no kernel de um sistema operacional é essencial para garantir a eficiência e o desempenho do sistema. Existem duas principais estratégias para alocar memória do kernel: o sistema de pares (sistema buddy) e a alocação de slabs.

O sistema de pares funciona alocando memória a partir de um segmento de tamanho fixo, composto por páginas fisicamente contíguas. A alocação é feita em unidades dimensionadas como potências de 2 (por exemplo, 4 KB, 8 KB, 16 KB). Isso pode resultar em fragmentação interna, onde uma solicitação de tamanho não exato é arredondada para o próximo tamanho de potência de 2 mais alto, levando a algum desperdício de memória.

Por outro lado, a alocação de slabs envolve agrupar objetos do kernel em caches, onde cada cache contém instâncias de uma estrutura de dados específica do kernel. Cada cache é preenchido com slabs, que são compostos por uma ou mais páginas fisicamente contíguas. Esse método evita desperdício de memória devido à fragmentação, pois cada objeto do kernel é alocado exatamente do tamanho necessário, sem arredondamento para potências de 2.

O alocador de slabs oferece benefícios como eliminação de desperdício de memória devido à fragmentação e rápida alocação de memória, sendo eficaz especialmente para sistemas onde objetos são alocados e desalocados frequentemente. No Linux, o alocador de slabs substituiu o sistema de pares como o método padrão de alocação de memória do kernel a partir da versão 2.2, proporcionando melhor desempenho e gerenciamento de memória.

O texto oferece uma visão detalhada sobre os discos magnéticos, destacando sua importância como uma parte crucial do armazenamento secundário em sistemas de computação contemporâneos. Ele descreve a estrutura física dos discos, incluindo os pratos circulares revestidos com material magnético, os cabeçotes de leitura-gravação e os cilindros compostos por trilhas e setores. Além disso, aborda o desempenho dos discos, explicando a taxa de transferência de dados, o tempo de busca e a latência rotacional. Também menciona os riscos associados ao contato entre o cabeçote e a superfície do disco, destacando o potencial de danos permanentes, conhecidos como "choque do cabeçote". Por fim, observa que alguns discos magnéticos são removíveis, permitindo a troca. Oferece uma análise abrangente sobre diversos tipos de dispositivos de armazenamento removíveis, como CDs, DVDs, discos Blu-ray e unidades flash, além de discutir a tecnologia de discos de estado sólido (SSDs) e fitas magnéticas. Ele explora a estrutura física e o funcionamento dos discos magnéticos, assim como sua conexão com computadores por meio de diferentes barramentos de entrada/saída.

Também são discutidas as vantagens e desvantagens dos SSDs em relação aos discos rígidos tradicionais, incluindo maior confiabilidade, velocidade e eficiência energética, mas custos mais elevados e capacidades limitadas. O texto ainda aborda o uso dos SSDs em arrays de armazenamento e laptops, bem como seu impacto no design de computadores. Por fim, menciona-se a fita magnética como uma mídia de armazenamento secundário mais antiga, ressaltando sua capacidade de retenção de dados, mas sua lentidão no acesso e baixa utilidade para armazenamento secundário rápido. A explorar diversos algoritmos de scheduling de disco, como FCFS, SSTF, SCAN, C-SCAN e LOOK, que visam otimizar o tempo de acesso e a largura de banda do disco ao ordenar solicitações de I/O.

Cada algoritmo possui vantagens e desvantagens, com o SSTF sendo comum devido ao seu desempenho superior em relação ao FCFS, enquanto SCAN e C-SCAN são mais adequados para sistemas com cargas pesadas de disco. A escolha do algoritmo depende do contexto do sistema e das características das solicitações de I/O, com a eficácia sendo influenciada pelo número e tipos de solicitações. O texto também aborda diversos aspectos do gerenciamento de discos e scheduling de disco. Ele destaca a influência do método de alocação de arquivos nas solicitações de serviço em disco, bem como a importância de algoritmos de scheduling de disco como módulos separados do sistema operacional para facilitar substituições quando necessário. Além disso, discute as complexidades dos SSDs e a necessidade de considerar não apenas as distâncias de busca, mas também a latência rotacional em discos modernos. O texto também menciona a implementação de algoritmos de scheduling de disco no hardware do controlador interno e a capacidade do sistema operacional de fazer seu próprio scheduling para atender a requisitos específicos. Por fim, explora aspectos do gerenciamento de discos, incluindo formatação, particionamento, criação de sistema de arquivos e o papel do bloco de inicialização no processo de inicialização do sistema operacional.

Ele fornece uma visão abrangente sobre diversos aspectos do gerenciamento de discos e memória em sistemas operacionais. Começando pela descrição do programa bootstrap completo em comparação com o carregador de bootstrap na ROM de inicialização, ele explora como o sistema operacional é inicializado a partir do disco, destacando o papel crucial do registro mestre de inicialização (MBR) no processo de inicialização do Windows. Além disso, o texto discute estratégias para lidar com blocos danificados em discos rígidos, como a reserva ou o deslocamento de setores, enfatizando a importância do gerenciamento eficiente do espaço de armazenamento para garantir a integridade dos dados. A análise do espaço de permuta revela a sua função na expansão da memória física através do uso do disco, detalhando diferentes abordagens adotadas por sistemas operacionais como Solaris e Linux para gerenciar esse espaço de forma eficiente.

Por fim, o texto introduz a estrutura RAID como uma técnica para melhorar tanto o desempenho quanto a confiabilidade do armazenamento de dados, através da distribuição de dados em múltiplos discos. No geral, o texto fornece uma compreensão abrangente dos desafios e estratégias envolvidos no gerenciamento eficiente de discos e memória em sistemas operacionais, destacando a importância desses aspectos para o funcionamento adequado do sistema e a integridade dos dados armazenados.

O sistema de arquivos é uma parte crucial e visível de um sistema operacional, fornecendo mecanismos para armazenamento e acesso a dados e programas. Ele é composto por uma coleção de arquivos e uma estrutura de diretórios que organiza e fornece informações sobre esses arquivos. Os arquivos, que são unidades lógicas de armazenamento, são mapeados para dispositivos físicos, como discos magnéticos e ópticos, que geralmente não são voláteis, garantindo a persistência dos dados entre reinicializações do sistema.

Um arquivo é uma coleção nomeada de informações relacionadas, armazenadas em memória secundária, e pode variar em estrutura, desde arquivos de texto até executáveis. Os atributos dos arquivos incluem o nome, um identificador único, o tipo, a localização no dispositivo de armazenamento, o tamanho, as permissões de acesso, e metadados como a hora e data de criação e modificação. Esses atributos ajudam a gerenciar e controlar o acesso aos arquivos, especialmente em sistemas com múltiplos usuários.

As operações básicas com arquivos envolvem criação, gravação, leitura, reposicionamento, exclusão e truncamento. A criação de um arquivo requer encontrar espaço no sistema de arquivos e criar uma entrada no diretório correspondente. Para gravar ou ler um arquivo, o sistema operacional utiliza ponteiros que indicam a posição no arquivo onde a operação deve ocorrer. Reposicionar um arquivo envolve alterar esse ponteiro sem necessidade de I/O real. Excluir um arquivo libera o espaço ocupado por ele e remove sua entrada no diretório. Truncar um arquivo redefine seu tamanho para zero, mantendo seus atributos intactos.

Os diretórios são estruturas que armazenam informações sobre todos os arquivos, incluindo o nome e identificador único de cada arquivo. Isso facilita a organização e o acesso aos arquivos no sistema. A gestão de arquivos abertos é feita por tabelas específicas que mantêm informações sobre os arquivos em uso, otimizando as operações de entrada e saída (I/O).

A questão do trancamento de arquivos é abordada com a implementação de locks, que podem ser compartilhados ou exclusivos. Esses locks são importantes em ambientes onde vários processos podem acessar o mesmo arquivo simultaneamente. O trancamento pode ser obrigatório, como no Windows, onde o sistema operacional impede o acesso concorrente, ou aconselhável, como nos sistemas UNIX, onde cabe ao software garantir a integridade do trancamento.

Em resumo, o sistema de arquivos é uma parte vital do sistema operacional, facilitando o armazenamento, acesso, e gerenciamento de dados e programas. Ele envolve a organização lógica dos arquivos, o gerenciamento de suas operações básicas, e a implementação de mecanismos para controle de acesso e concorrência. O texto oferece uma visão abrangente desses aspectos, detalhando como o sistema operacional lida com a criação, manipulação e proteção dos arquivos. Para um entendimento mais prático, exemplos concretos de operações de arquivo em diferentes sistemas operacionais poderiam ser incluídos, ajudando a ilustrar melhor esses conceitos.

Sistemas de arquivos são um dos aspectos mais visíveis de um sistema operacional para os usuários. O sistema de arquivos fornece os mecanismos necessários para o armazenamento e o acesso a dados e programas, tanto para o sistema operacional quanto para todos os usuários de um computador. Ele é composto por duas partes principais: uma coleção de arquivos, onde cada arquivo armazena dados relacionados, e uma estrutura de diretórios, que organiza e fornece informações sobre todos os arquivos no sistema. Além de armazenar dados, o sistema de arquivos lida com a organização desses dados, permitindo que os usuários encontrem e acessem informações de forma intuitiva. A estrutura de diretórios desempenha um papel essencial nesse contexto, ajudando a classificar e localizar arquivos.

Sobre a semântica de compartilhamento de arquivos, que é fundamental em ambientes onde múltiplos processos, usuários e computadores precisam acessar os mesmos arquivos. Essa parte do sistema de arquivos deve garantir que o compartilhamento ocorra de maneira segura e consistente, evitando conflitos e garantindo a integridade dos dados.

Os computadores utilizam diversas mídias de armazenamento, como discos magnéticos, fitas magnéticas e discos óticos, para guardar informações. Para simplificar o uso dessas informações, o sistema operacional fornece uma visão lógica uniforme das mesmas, abstraindo as características físicas dos dispositivos de armazenamento e definindo a unidade lógica de armazenamento conhecida como arquivo. Esses arquivos são gerenciados pelo sistema operacional, que os mapeia para dispositivos físicos, geralmente não voláteis, garantindo que os dados persistam entre reinicializações do sistema.

Um arquivo é simplificadamente uma coleção nomeada de informações relacionadas, armazenadas em memória secundária. Do ponto de vista do usuário, o arquivo é a menor unidade de armazenamento lógico secundário; dados não podem ser gravados na memória secundária sem estarem contidos em um arquivo. Os arquivos podem representar tanto programas (nas formas fonte e objeto) quanto dados, que podem ser numéricos, alfabéticos, alfanuméricos ou binários. Eles podem ter uma estrutura livre, como arquivos de texto, ou serem rigidamente formatados. Em geral, um arquivo é uma sequência de bits, bytes, linhas ou registros, cujo significado é definido pelo seu criador e usuário, tornando o conceito de arquivo bastante abrangente.

As informações contidas em um arquivo são definidas por seu criador, possibilitando o armazenamento de diversos tipos de dados, como programas-fonte ou executáveis, dados numéricos, textos, fotos, músicas e vídeos. A estrutura de um arquivo é específica e dependente do seu tipo: um arquivo de texto é organizado em linhas; um arquivo-fonte é composto por funções organizadas como declarações seguidas por comandos executáveis; e um arquivo executável é constituído por seções de código que podem ser carregadas na memória e executadas pelo sistema.

Esse detalhamento mostra como os sistemas operacionais tratam os arquivos de maneira geral, possibilitando a manipulação eficiente e segura de diferentes tipos de informações e garantindo a persistência e integridade dos dados armazenados. Um arquivo é nomeado para facilitar o uso pelos usuários humanos, sendo referenciado pelo seu nome, que geralmente é uma sequência de caracteres, como "example.c".

Alguns sistemas operacionais diferenciam maiúsculas e minúsculas nos nomes dos arquivos, enquanto outros não. Quando um arquivo recebe um nome, ele se torna independente do processo, usuário ou sistema que o criou. Por exemplo, um arquivo chamado "example.c" pode ser criado por um usuário e editado por outro, transferido via USB, enviado por e-mail ou copiado por meio de uma rede, mantendo o mesmo nome no sistema de destino.

Os atributos de um arquivo variam entre sistemas operacionais, mas geralmente incluem:

1. Nome: O Nome Simbólico do arquivo, legível por humanos.

2. Identificador: Um rótulo exclusivo, geralmente numérico, que identifica o arquivo dentro do sistema de arquivos, não legível por humanos.

3. Tipo: Informação necessária para sistemas que suportam diferentes tipos de arquivos.

4. Localização: Um ponteiro para o dispositivo e a localização do arquivo no dispositivo.

5. Tamanho: O tamanho atual do arquivo (em bytes, palavras ou blocos) e possivelmente o tamanho máximo permitido.

6. Proteção: Informações de controle de acesso que determinam quem pode ler, gravar, executar o arquivo, etc.

7. Hora, data e identificação do usuário: Informações sobre a criação, última modificação e última utilização do arquivo, úteis para proteção, segurança e monitoramento do uso.

Alguns sistemas de arquivos modernos também suportam atributos de arquivo estendidos, como a codificação de caracteres e recursos de segurança, como a soma de verificação do arquivo.

As informações sobre todos os arquivos são mantidas na estrutura de diretório, que também reside na memória secundária. Uma entrada no diretório normalmente contém o nome do arquivo e seu identificador exclusivo, que localiza os outros atributos do arquivo. O registro dessas informações para cada arquivo pode ocupar mais de um kilobyte, e em sistemas com muitos arquivos, o tamanho do diretório pode chegar a megabytes. Como os diretórios devem ser não voláteis, eles são armazenados no dispositivo de memória secundária e trazidos para a memória conforme necessário.

Um arquivo é um tipo de dado abstrato que requer operações específicas para ser manipulado pelo sistema operacional. As operações básicas incluem:

1. Criação de um arquivo: Envolve encontrar espaço no sistema de arquivos e criar uma entrada no diretório.

2. Gravação em um arquivo: Requer especificar o nome do arquivo e os dados a serem gravados, atualizando um ponteiro de gravação.

3. Leitura de um arquivo: Requer especificar o nome do arquivo e onde armazenar os dados lidos, atualizando um ponteiro de leitura.

4. Reposicionamento dentro de um arquivo: Ajusta o ponteiro da posição corrente do arquivo sem realizar I/O real.

5. Exclusão de um arquivo: Libera o espaço ocupado pelo arquivo e remove sua entrada no diretório.

6. Truncamento de um arquivo: Apaga o conteúdo do arquivo, mantendo seus atributos, mas redefinindo seu tamanho para zero.

Essas operações são essenciais e formam o conjunto mínimo de operações de arquivo. Operações adicionais, como renomear um arquivo ou adicionar dados ao final de um arquivo existente, podem ser implementadas combinando as operações básicas. Para otimizar o acesso aos arquivos, muitos sistemas utilizam a chamada de sistema open(), que abre um arquivo e armazena suas informações em uma tabela de arquivos abertos. Isso evita buscas constantes no diretório. O sistema pode manter tabelas de arquivos abertos tanto por processo quanto por todo o sistema, controlando ponteiros de posição, contagens de aberturas e localizações de arquivos em disco.

Além disso, alguns sistemas operacionais oferecem trancamento de arquivos para evitar acessos simultâneos por múltiplos processos. Locks podem ser compartilhados (permitindo acesso concorrente) ou exclusivos (permitindo acesso por apenas um processo de cada vez). Locks podem ser obrigatórios, onde o sistema operacional impede acessos concorrentes automaticamente, ou aconselháveis, onde cabe aos desenvolvedores implementar o uso adequado dos locks.

Por fim, a implementação cuidadosa do trancamento é necessária para evitar deadlocks e garantir a integridade dos dados em sistemas com múltiplos processos acessando os mesmos arquivos.