Notas de Aula 13 - Algoritmos e Estrutura de Dados 2 (AE43CP) – Pesquisa em Memória Secundária
Prof. Jefferson T. Oliva

Pesquisa em memória primária
- Pesquisa sequencial
- Pesquisa binária
- Árvores de pesquisa
→ Binária
→ AVL
→ Vermelha-preta
→ Árvores B
- Hashing

Pesquisa em memória secundária envolve mais registros do que a memória interna pode suportar
- Custo para acessar registro é maior: minimizar acessos, ou seja, transferência da memória secundária para memória primária.
- Métodos eficientes de pesquisa dependem das características de hardware e sistema operacional.
- Medida de complexidade: custo de transferir dados entre a memória principal e secundária (minimizar o número de transferências).
- Apenas um registro pode ser acessado por vez.
→ Fitas magnéticas: acesso de forma sequencial.
→ Discos: acesso direto, mas todo bloco deve ser trazido à memória.

Modelo Computacional para Memória Secundária

O modelo utilizado para a memória secundária é a memória virtual, uma técnica que usa a memória secundária como uma cache (cache é um dispositivo de acesso rápido, interno a um sistema, que serve de intermediário entre um operador de um processo e o dispositivo de armazenamento) para armazenamento secundário [Wikipedia]. A memória virtual, utiliza uma técnica sofisticada de gerenciamento de memória, onde as memórias principal e secundária são combinadas, dando a ilusão de existir uma memória maior que a memória principal. Ou seja, a memória virtual é uma técnica que utiliza parte da memória secundária como se fosse uma memória RAM, assim o sistema operacional executa aplicações mesmo que os programas em execução supere a capacidade da memória.
- Normalmente implementado como uma função do sistema operacional
- Uso de uma pequena quantidade de memória principal e uma grande quantidade de memória secundária
- Boa estratégia para algoritmos com pequena localidade de referência, que é um conceito vindo de arquitetura de computadores, como a memória do computador é finita, existe alguns algoritmos que visam deixar o acesso a memória mais rápidos. A ideia principal é que existe memórias que são mais rápidas que outras, por exemplo, é muito mais rápido acessar um conteúdo na memória RAM do que no disco. Imagine que você tem uma parte do seus dados que é muito acessada, e outras que não são tanto, e que todos eles estão no disco, toda vez que você ler ou escreve esse dado, você

precisará ler e escrever no disco, envolvendo operações de I/O, que são demoradas. O ideal seria que os dados que são mais acessados estivessem na memória RAM.
- Organização do fluxo entre a memória principal e secundária é extremamente importante.
- Programador pode endereçar grandes quantidades de dados, deixando para o sistema a responsabilidade de transferir o dado da memória secundária para a principal.

Memória virtual: funções básicas
- Realocação: para assegurar que cada processo (aplicação) tenha o seu próprio espaço de endereçamento, começando em zero.
- Proteção: para impedir que um processo utilize um endereço de memória que não lhe pertença.
- Paginação: (paging) ou troca (swapping), que possibilita a uma aplicação utilizar mais memória do que a fisicamente existente (essa é a função mais conhecida).

Sistema de paginação: paginação é um esquema de gerenciamento de memória pelo qual um computador armazena e recupera dados de um armazenamento secundário para uso na memória principal. O mapeamento de endereços permite o programador utilizar um espaçamento de endereço maior que o espaço de memória primária disponível.
- Implementação mais utilizada.
- Espaço de endereçamento dividido em páginas de igual amanho (geralmente múltiplos de 512 bytes).
- Memória principal dividida em molduras de páginas (frames) de igual tamanho que são pequenas partições. Cada página é mapeada em um frame de memória através de um processo que chama paginação.
- Molduras de páginas contêm algumas páginas ativas.
- Páginas inativas estão na memória secundária.

Mecanismo de paginação tem duas funções:
1 - Mapeamento dos endereços: determinar qual página está sendo usada e encontrar a moldura (se existir).
2 - Transferência das páginas entre memória primária e secundária.

Referência de página
- Uma parte dos bits do endereço é definida como um número de página
- Outra parte é o número do byte do item dentro da página
- Exemplo: se o espaço de endereçamento é de 24 bits
→ A memória terá virtual terá 2^24 bytes
→ Se a página é de 512 bytes (2^9): nove bits são usados para o número do byte dentro da página (b) e o restante (15 bits) representa o número da página (p)

O mapeamento de endereços das memórias principal e secundária é realizado por meio de uma Tabela de Páginas que tem um tamanho fixo: geralmente menor que o número de páginas
- f(p, b) = p’ + b
- A p-ésima entrada da tabela contém a localização p' da moldura de página (frame) que contém a página número p, caso a mesma esteja na memória principal

- A tabela de páginas pode ser representada como um vetor do tamanho de número possível de páginas
- Se a página número p não estiver em um frame na memória principal, p' terá valor nulo

Ver slide 11

Caso um programa necessitar de uma página que não esteja na memória principal (p = NULL), essa página p deve ser recuperada da memória secundária para a primária.
- Também, a tabela de páginas deve ser atualizada.

Se não houver uma moldura de página vazia, uma página deverá ser removida da memória principal
- O ideal é a remoção da página que não será referenciada pelo período de tempo mais longo no futuro.
- Como não há como prever o futuro, o mesmo é inferido a partir do comportamento passado.

Algoritmos para escolha da página a ser removida:
- Menos recentemente utilizada (LRU - "least recently used").
- Menos frequentemente utilizada (LFU - "least frequently used").
- Ordem de chegada (FIFO - first in first out).

Menos recentemente utilizada (LRU)
- Um dos algoritmos mais utilizados.
- Remove a página menos recentemente utilizada.
- Princípio: comportamento futuro deve seguir o passado recente.
- Implementação por meio de filas
→ Após a utilização de uma página, ela é colocada no fim da fila.
→ A página no início da fila é que foi menos recentemente utilizada.
→ A nova página trazida da memória secundária deve ser colocada no frame que contém a página menos recentemente utilizada.

Menos frequentemente utilizada (LFU)
- Remove a página menos frequentemente utilizada.
- Custo extra: registrar número de acessos a cada página.
- Desvantagem: uma página recentemente trazida da memória secundária tem um baixo número de acessos e pode ser brevemente removida.

Ordem de chegada (FIFO)
- Remove a página que está na memória principal há mais tempo
- Algoritmo mais simples e barato de se manter
- Desvantagem: ignora que a página mais antiga pode ser a mais referenciada

Antes de implementarmos um exemplo de método de busca em memória secundária, iremos ver operações com arquivos

Exemplo de Implementação com Paginação

Podemos simular a memória virtual por meio de arquivos binários (.dat).

Também, parte das informações podem estar na memória principal (e.g. chave e posição de sua respectiva página) e o restante na secundária (e.g. demais dados de um registro).

Exemplo:

```
#define ITENSPAGINA 5
#define MAXTABELA 20

typedef struct{
	long RA;
	char nome[70];
	int codigo_curso;
	float coef;
}Aluno;

typedef struct{
	Aluno itens[ITENSPAGINA];
}Pagina;


int buscar(Aluno* aluno, FILE *arq){
	Pagina pag;
	int i = 0;
	int tam = tamanho_arquivo(arq);

	fseek(arq, 0, SEEK_SET);

	if (tam > 0){
		do{
			fread(pag, sizeof(Pagina), 1, arq);

			for (i = 0; (i < ITENSPAGINA) && (aluno->RA < pag->itens[i]->RA); j++);

			if ((i < ITENSPAGINA) && (aluno->RA == pag->itens[i]->RA)){
				*aluno = pag->itens[i];
				return 1;
			}else
				fseek(arq, sizeof(Pagina), SEEK_CUR);
		}while (ftell(arq) < tam);
	}
	return 0;
}


int tamanho_arquivo(FILE *arq){
	fseek(arq, 0, SEEK_END);
	return ftell(arq);
}
```

Complexidade:
- Carregar M páginas: O(M)
- Percorrer N itens em cada página: O(N)
- Custo total: O(M * N)

Acesso Sequencial Indexado

Utiliza o princípio da pesquisa sequencial: cada item é lido sequencialmente até encontrar uma chave maior ou igual a chave de pesquisa

Providências necessárias para aumentar a eficiência da pesquisa sequencial:
- O arquivo deve estar ordenado pelo campo chave do item
- Um arquivo de índice de páginas, contendo pares de valores (x, p), deve ser criado, onde
→ x é a chave de um item
→ p é o endereço da página na qual o primeiro item contém a chave x

Exemplo para um conjunto de 15 registros (itens)
- Cada página tem capacidade para armazenar 4 itens do disco
- Cada entrada do índice de páginas armazena a chave do 1º item de cada página e o endereço de tal página no disco

Para se pesquisar por um item, deve-se:
- Localizar, no índice de páginas, a página que pode conter o item desejado de acordo com sua chave de pesquisa
- Realizar uma pesquisa sequencial na página localizada

Complexidade:
- Percorrer a tabela de índice de tamanho m
- Percorrer a página de tamanho n
- Custo total: O(n + m)

Figuras nos slides entre 27 e 30 (link do exemplo no youtube)

Várias superfícies de gravação são utilizadas em um disco magnético.

O disco é dividido em vários círculos concêntricos (trilhas). Nos discos, as trilhas estão verticalmente alinhadas, formando um cilindro.

Cilindro: todas as trilhas verticalmente alinhadas e que possuem o mesmo diâmetro.

Acesso sequencial indexado em disco magnético considera:
- Latência rotacional: tempo necessário para que o início do bloco contendo o registro a ser lido passe pela cabeça de leitura/gravação.

Tempo de busca (seek time): tempo necessário para que o mecanismo de acesso desloque de uma trilha para outra (maior parte do custo para acessar dados)

Acesso sequencial indexado em disco magnético para localizar um registro:
- Localização do cilindro correspondente no índice de cilindros
- Deslocamento da cabeça de gravação até o cilindro
- Leitura da página que contém índice de páginas
- Leitura página de dados com a chave procurada
- Possibilita acesso sequencial ou randômico
- Adequado apenas para aplicações com baixa frequência de inserção e remoção
- Vantagem: garantia de acesso com apenas um deslocamento da cabeça de gravação

Desvantagem: inflexibilidade, pois, se há muita inserção e remoção, dados tem que ser reorganizados

Referências

Assis, G. T. Pesquisa Externa. BCC203 - Algoritmos II. Notas de Aula. Ciência da Computação. DECOM/UFOP, 2018.

Deitel, H. M. e Deitel, P. J. C: Como Programar. Pearson, 2011.

Schildildt, H. C Completo e Total. Pearson, 2011.

Ziviani, N. Projeto de Algoritmos - com implementações em Java e C++. Thomson, 2007.