

Fundamentos de Arquivos

SCC 503 - Alg. Estrut. de Dados II

Sumário

1. Introdução

- a. Características do armazenamento secundário e disco
- b. Estruturas de arquivos: objetivos e exemplos
- c. Histórico

2. Operações e fundamentos de arquivos

- a. Arquivos Físicos e Lógicos
- b. Implementação em C
- c. Comandos do sistema de arquivos UNIX

Arquivos

Informação mantida em memória secundária

- HD
- Disquetes (é do arco da velha !)
- Fitas magnéticas
- CD, DVD e Blu-ray
- Memória flash: pen-drives, mp3-9 player, cartões
- Outros?
- Futuro?
 - bactérias (<http://geekbeat.tv/bacteria/>)
 - átomos,
 - cristais
 - <https://www.crn.com/news/storage/217200230/ge-unveils-500-gb-holographic-disc-storage-technology.htm>

Terminologia

- **arquivo** : uma estrutura de dados em um sistema de arquivos, que é mapeado para nomes para objetos como arquivos ou diretórios.
- **estrutura de arquivo** (*file structure*): um padrão para se organizar dados num arquivo (incluindo ler, escrever e modificar).
- **algoritmo** : um conjunto finito de regras bem-definidas para a solução de um problema num número fininito de passos.
- **estrutura de dados** : um padrão para organizar dados num algoritmo ou programa.

Terminologia

- **armazenamento volátil** : armazenamento que perde o conteúdo quando não alimentado por energia.
- **armazenamento não-volátil** : armazenamento que retém o conteúdo quando não alimentado por energia.
- **dados persistentes** : informação que é retida mesmo após a execução de um programa que a cria.

Velocidade de acesso

Discos são lentos! (assim como outros dispositivos para armazenamento secundário).

- No entanto são muito úteis por combinar baixo custo, alta capacidade de armazenamento e portabilidade.

Quão lentos?

- O tempo de acesso típico em memória principal (RAM) é de nanossegundos, ou 0,00007 milissegundos ≈ 70 (já existe tecnologia para acesso em até 10 ns) .
- Para acessar a mesma informação em disco, ≈ 10 milissegundos (já existem discos com acesso a 8 ms) .
- uma diferença da ordem de 140 . 000 (em configurações típicas atuais, mas essa diferença pode variar entre 100 mil e 300 mil) .

Velocidade de Acesso

Uma analogia para entender a diferença na velocidade do acesso:

- se acessar a memória principal é como encontrar uma informação num livro usando o índice desse livro, em 20 segundos,
- acessar o disco seria como ter que fazer uma requisição a uma bibliotecária para que ela procure uma informação numa biblioteca.
- **comparativamente, usando o cálculo anterior, significa que obter a informação demoraria por volta de 777.8 horas, ou pouco mais de 32 dias !!**

Outras comparações

Custo:

- RAM : US\$ 0.05 / Mbit
HD : US\$ 0.0002 / Mbit

Capacidade:

- RAM : acima de 1 Gigabyte
HD : acima de 1 Terabyte

Volatilidade:

- RAM : volátil
HD : não-volátil

Persistência:

- RAM : informação é retida enquanto o programa que controla as variáveis estiver sendo executado.
HD : informação pode ser persistente.

Objetivos no estudo de estruturas de arquivo

- Minimizar o número de acessos ao disco : idealmente obter/processar a informação num único acesso
- Maximizar a quantidade de informações recuperadas ou processadas em um acesso, agrupando informações relacionadas.
- De forma independente da tecnologia : $\text{Tempo de acesso} = \text{Número de acessos} \times \text{tempo de 1 acesso}$.
- Deve-se ter cautela para não projetar uma estrutura de arquivo muito dependente da tecnologia atual (há 10-15 anos o uso de disquete e CD era amplo e hoje quase inexistente)

Estruturas de Arquivo

- **Estruturas de dados eficientes em memória são muitas vezes inviáveis em disco.**
- Um dos problemas em se obter uma estrutura de dados adequada é a constante necessidade de alterações em arquivos.
- O ideal é evitar sequências de acessos (várias requisições à bibliotecária, no exemplo anterior).

Um exemplo de inviabilidade

O método de busca binária permite que 1 registro pesquisado entre 50 mil seja encontrado em no máximo 16 comparações ($\log_2 (50000) \approx 16$)

- Mas acessar o disco 16 vezes é demais.
- É preciso um método que recupere esse mesmo registro em poucos acessos.
 - Agrupar informações para permitir recuperar o máximo de informação em uma única operação de acesso
 - Por exemplo, ao consultar um pedido de um cliente e buscar suas informações pessoais (nome, endereço, telefone, CPF, etc.), é preferível obter todas as informações de uma vez ao invés de procurar em vários lugares

Histórico

Os primeiros trabalhos com arquivos presumiam o armazenamento em fita

- acesso sequencial
- tamanho dos arquivos cresceu muito e inviabilizou esse tipo de acesso

São, no entanto ainda usadas principalmente para armazenamento off-line redundante pela vida útil longa que apresentam (até 100 anos)

- vide caso de recuperação de perdas do Gmail por backup em fitas.
 - <https://www.geek.com/news/google-restoring-lost-gmail-data-from-tape-backups-1320631/>



Histórico

Em discos

- foram adicionados índices aos arquivos que tornaram possível manter uma lista de chaves e ponteiros para acesso aleatório.
- mais uma vez o crescimento dos arquivos de índice tornou difícil a sua manutenção.

Em 1960 o uso de árvores surgiu como solução em potencial, com a desvantagem de crescerem de maneira desigual

- Árvores AVL (1963) foram investigadas para esse problema.
- Árvores-B : demorou 10 anos para serem desenvolvidas pela diferença de abordagem em disco e RAM crescimento bottom-up e acesso sequencial as tornaram a base para os sistemas de arquivos.
- Hashing : possibilita acesso em tempo constante, mas em arquivos que não se modiquem.
- Hashing dinâmico : permitiu modificação no tamanho dos índices e recuperação de informação em no máximo dois acessos.

Arquivos Físicos e Lógicos

Um arquivo sempre é físico do ponto de vista do armazenamento. É um conjunto de bytes armazenados e rotulados com um nome.

- Para um aplicativo a noção é diferente, pois só é possível acompanhar o fluxo de bytes de leitura e escrita no arquivo.
- O programa é geralmente limitado a 20 conexões com arquivos (analogia com linhas telefônicas).
- Os bytes podem ser originários de um arquivo físico, do teclado ou outros dispositivos.

Assim, a linha de comunicação aberta pelo sistema operacional para o programa é chamado de arquivo lógico, o que é distinto do arquivo físico no disco ou fita, gerenciado apenas pelo sistema operacional.

Arquivos Físicos e Lógicos

No código fonte, uma instrução liga o arquivo físico a uma variável lógica. Uma vez ligado, é preciso declarar o que desejamos executar no arquivo.

Em geral duas opções:

- abrir um arquivo existente, ou
- criar um novo arquivo, apagando qualquer conteúdo anterior no arquivo físico.

Após abrir um arquivo estaremos posicionados no início do arquivo e, portanto, prontos para escrever ou ler.

Implementação em C

Dois níveis de manipulação

- funções em baixo nível
- funções de alto nível:
 - implementadas em baixo
 - mantém área de memória (buffer) para manipulação dos bytes

Implementação em C

Na stdio.h :

```
#define FOPEN_MAX (20)
```

```
typedef struct _iobuf
```

```
{
```

```
    char* _ptr;
```

```
    int _cnt;
```

```
    char* _base;
```

```
    int _flag;
```

```
    int _file;
```

```
    int _charbuf;
```

```
    int _bufsiz;
```

```
    char* _tmpfname;
```

```
} FILE;
```

Implementação em C

Abertura de arquivo:

```
FILE *fd=fopen(<filename>,<flags>)
```

- filename : nome do arquivo a ser aberto
- flags : modo de abertura
 - r : apenas leitura (o arquivo precisa existir)
 - w : cria arquivo vazio para escrita (apaga um arquivo já existente)
 - a : adiciona conteúdo a um arquivo
 - r+ : abre arquivo para leitura e escrita
 - w+ : cria arquivo vazio para leitura e escrita
 - a+ : abre arquivo para leitura e adição de dados
 - b : inserir após as ags anteriores para trabalhar com arquivo binário, caso contrário será aberto em modo texto.

Implementação em C

Fechamento de arquivo, transfere o restante da informação no buffer e desliga a conexão com o arquivo físico:

```
fclose(<fd>)
```

- `fd` : *file descriptor*, do tipo ponteiro FILE

Porque se utiliza *buffer* ?

Implementação em C

Grupos de funções para manipulação de arquivos:

- por caractere
- por cadeia de caracteres
- dados formatados
- blocos de bytes

Implementação em C

Por caractere

- `fputc(<char>,<FILE>)` : escreve um caractere no arquivo,
- `<char> = fgetc(<FILE>)` : lê um caractere do arquivo.

EOF : caractere que indica o fim de arquivo. `feof(<FILE>)` : função que retorna 1 se fim de arquivo

Implementação em C

Por cadeia de caracteres

- `fputs(<char *>, <FILE>)` : escreve uma string no arquivo,
- `fgets(<char *>, <int>, <FILE>)` : lê do arquivo uma determinada quantidade de caracteres e armazena a cadeia de caracteres numa variável, retorna NULL se fim de arquivo.

Implementação em C

Por dados formatados

- `fprintf(<FILE>,"formatacao", ...)` : similar ao `printf`, escreve num arquivo a string formatada.
- `fscanf(<FILE>,"formatacao", ...)` : similar ao `scanf`, arquivo strings formatadas, retorna EOF se fim de arquivo.

Implementação em C

Por blocos de bytes (arquivos binários), assim como estão armazenados na memória principal

- `<size_read> = fread(<buffer>, <size_un>, <size_buffer>, <FILE>)` :
 - `size_read` : unidades lidas (0 se m de arquivo),
 - `buffer` : variavel que vai armazenar a leitura,
 - `size_un` : tamanho de cada unidade (bloco) de bytes a ser lido,
 - `size_buffer` : número de blocos
 - `FILE` : ponteiro FILE
- `fwrite(<buffer>, <size_un>, <size_read>, <FILE>)` : similar ao `scanf` , lê do arquivo strings formatadas, retorna EOF se fim de arquivo.
- `fseek(<FILE>, <move_bytes>, <start_byte>)` : move o ponteiro do arquivo para uma posição determinada.

Redirecionamentos de E/S e Pipes em Unix

O redirecionamento de entrada e saída permite especificar a gravação ou leitura de um arquivo em substituição à entrada: stdin e saída: stdout .

- `< file` : redireciona stdin para “file”
- `> file` : redireciona stdout para “file”

Pipes permitem o envio de saídas de um programa para serem usados por outros programas

- `programa1 | programa 2` : pega a saída stdout do programa1 e usa na entrada stdin do programa2.
- Um exemplo: `ls | sort` : a listagem de arquivos é enviada para o programa sort, que as ordena e mostra na tela

Outros Comandos

- `cat <filename1> <filename2>...` : lista o conteúdo dos arquivos
- `tail <filename>` : lista as dez últimas linhas do arquivo
- `cp <filename1> <filename2>` : copia o arquivo1 para o arquivo2
- `mv <filename1> <filename2>` : move (e renomeia) o arquivo1 para o arquivo2
- `rm <filename>` : apaga arquivos
- `ls` : lista o conteúdo do diretório atual
- `mkdir <name>` : cria um diretório
- `rmdir <name>` : remove um diretório
- `cd <name>` : acessa um diretório