Fundamentos de Arquivos

SCC0607 – Estrutura de Dados III

Anderson Canale Garcia

Sumário

- Características do armazenamento secundário em disco
- Estruturas de arquivos
- Histórico
- Arquivos físicos e lógicos
- Implementação em C

Arquivos

Informação mantida em memória secundária

- Fitas magnéticas (obsoleto)
- Disquetes (obsoleto)
- CD, DVD e Blu-ray (obsoleto)
- Memória flash (pen-drives, cartões, etc.)
- HD
- Outros

Terminologia geral

- **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 modicar).
- **algoritmo:** um conjunto finito de regras bem-definidas para a solução de um problema num número finito de passos.
- **estrutura de dados:** um padrão para organizar dados num algoritmo ou programa.

Terminologia de acesso a arquivos

- **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).
  - Porém, combinam baixo custo, alta capacidade de armazenamento e portabilidade.
- Quão lentos?
  - O tempo de acesso típico em memória principal (RAM) é de ≈ 70 nanossegundos, ou 0,00007 milissegundos
    (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 - Analogia

- 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.

Velocidade de acesso – Outras comparações

	RAM	HD
Custo	US$ 0.05 / Mbit	US$ 0.0002 / Mbit
Capacidade	Acima de 1 Gigabyte	Acima de 1 Terabyte
Volatilidade	Volátil	Não-volátil
Persistência	Informação é retida enquanto o programa que controla as variáveis estiver sendo executado	Informação pode ser persistente

Objetivos no estudo de estrutura de arquivo

- Minimizar o número de acessos ao disco
  - Idealmente obter/processar a informação num único acesso
- Agrupar informações relacionadas
  - Maximizar a quantidade de informações recuperadas ou processadas em um acesso

- De forma independente da tecnologia:
  - Tempo de acesso = Número de acessos X tempo de 1 acesso.
- Deve-se ter cautela para não projetar uma estrutura de arquivo muito dependente da tecnologia atual
  (há 20 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).

Exemplo de inviabilidade

- Busca binária
  - Permite encontrar 1 registro entre 50 mil em 16 comparações
    - $\log_2(50000) \approx 16$
  - 16 acessos a disco?

- Alternativas
  - Agrupar informações
  - Exemplo: buscar informações de um cliente (nome, endereço, telefone, CPF, etc.)

Sumário

- Características do armazenamento secundário em disco
- Estruturas de arquivos
- **Histórico**
- Arquivos físicos e lógicos
- Implementação em C

Histórico

- Primeiros trabalhos com arquivos presumiam o armazenamento em fitas
  - Acesso sequencial
  - Aumento no tamanho dos arquivos inviabilizou esse tipo de acesso

- Ainda são usados para armazenamento offline redundante
  - Vida útil longa (até 100 anos)
  - https://exame.com/tecnologia/gmail-usa-fita-para-recuperar-contas/

Histórico

- Uso de discos
- Criação de índices aos arquivos
  - Lista de chaves e ponteiros para acesso aleatório
  - Crescimento dos arquivos de índice → dificuldade de manutenção

- Em 1960 → uso de árvores
  - Desvantagem: crescimento de maneira desigual
  - Possíveis soluções?

Sumário

- Características do armazenamento secundário em disco
- Estruturas de arquivos
- Histórico
- **Arquivos físicos e lógicos**
- Implementação em C

Arquivos Físicos e Lógicos

- Um arquivo sempre é físico do ponto de vista do armazenamento
  - Conjunto de bytes armazenados e rotulados com um nome

- Para um aplicativo, a noção é diferente
  - Fluxo de bytes de leitura e escrita no arquivo
  - Os bytes podem ser originários de um arquivo físico, do teclado ou outros dispositivos

- Exemplo

Arquivos Físicos e Lógicos

- Um arquivo sempre é físico do ponto de vista do armazenamento
  - Conjunto de bytes armazenados e rotulados com um nome

- Para um aplicativo, a noção é diferente
  - Fluxo de bytes de leitura e escrita no arquivo
  - Os bytes podem ser originários de um arquivo físico, do teclado ou outros dispositivos

- No código fonte, uma instrução liga o arquivo físico a uma variável lógica. Duas opções:
  - abrir um arquivo existente, ou
  - criar um novo arquivo, apagando qualquer conteúdo anterior no arquivo físico

Sumário

- Características do armazenamento secundário em disco
- Estruturas de arquivos
- Histórico
- Arquivos físicos e lógicos
- **Implementação em C**

Implementação em C | 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 do 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

fclose(<fd>)

- Fechamento de arquivo, transfere o restante da informação no bufer e desliga a conexão com o arquivo físico
  - **fd:** *file descriptor*, do tipo o ponteiro FILE

- Por que se utiliza *buffer*?

Implementação em C | Funções

Grupos de funções para manipulação de arquivos:

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

Implementação em C | Funções

- Por caractere

`fputc(<char>,<FILE>):` escreve um caractere no arquivo

`<char> = fgetc(<FILE>):` lê um caractere do arquivo

`EOF:` caractere que indica fim de arquivo.

`feof(<FILE>):` função que retorna 1 se fim de arquivo

Implementação em C | Funções

- Por cadeia de caracteres

`fputs(<char *>,<FILE>):` escreve uma caractere 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 m de arquivo.

Implementação em C | Funções

- Por dados formatados

`fprintf(<FILE>,"formatacao", ...)`: similar ao printf, escreve num arquivo a strinf formatada

`fscanf(<FILE>,"formatacao", ...)`: similar ao scanf, lê do arquivo strings formatadas, retorna EOF se fim do arquivo.

Implementação em C | Funções

- Por blocos de bytes (arquivos binários)
  - Assim como são 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>)`

`fseek(<FILE>,<move_bytes>,<start_byte>):` move o ponteiro do arquivo para uma posição determinada

Bibliografia

FOLK, M.J. et al
File Structures: an object-oriented approach with C++
Capítulos 1 e 2

FOLK, M.J. et al
File Structures
Capítulos 1 e 2

YOUNG, J.H.
File Structures
http://www.comsci.us/fs/notes/ch01a.htm