Algoritmos e Estruturas de Dados II - SCC-203

Arquivos: Fundamentos

Gustavo Batista

Arquivos

- Mecanismo de organização da informação mantida em memória secundária:
  - HD;
  - Disquete;
  - Fitas;
  - CD.
- Da mesma forma que variáveis e alocação dinâmica organizam o acesso do programa à memória principal.

Arquivos

- Existem dois principais motivos para a utilização de arquivos:
  1. Acesso a memória não volátil;
  2. Acesso a memória em grande quantidade e baixo custo.
- Entretanto, existe um grande custo associado:
  - Grande tempo de acesso.

Discos vs Memória Principal

- Tempo de acesso:
  - HD: alguns milisegundos ~ 10ms;
  - RAM: alguns nanosegundos ~ 10ns...40ns;
  - Ordem de grandeza da diferença entre os tempos de acesso é aproximadamente 250.000;
  - Porque? Basicamente porque discos são mecânicos e envolvem partes móveis.

Arquivos vs Memória Principal

	RAM	Discos (HD)	CDs
Custo	Alto	Baixo	Muito baixo
Tempo de Acesso	Baixo	Alto	Muito alto
Capacidade	Baixa	Alto	Médio
Princípio	Elétrico	Magnético	Ótico
Persistência	Volátil	Não-volátil	Não-volátil
Acesso	Aleatório	Aleatório	Parcialmente aleatório
Organização	Células	Trilhas/Setores	Trilhas/Setores

Organização de Arquivos

- Meta: criar estruturas para minimizar as desvantagens do uso da memória externa.
- Objetivo: minimizar o tempo de acesso ao dispositivo de armazenamento externo.

Organização de Arquivos

- Idealmente se deseja encontrar a informação em um único acesso.
- Se não for possível, deve-se encontrar em poucos acessos.
  - Mesmo uma busca binária não é eficiente o suficiente (~16 acessos para 50.000 registros).
- Deseja-se preferencialmente encontrar todas as informações necessárias em uma única leitura.

Organização de Arquivos

- Estruturas de dados eficientes em memória principal são inviáveis em memória secundária.
- Seria fácil obter uma estrutura de dados adequada para disco se os arquivos fossem estáticos (não sofressem alterações).

Organização de Arquivos

- Por exemplo, árvores AVL (Adelson-Velskii & Landis, 1962) são consideradas rápidas para pesquisa em memória principal, mas mesmo com árvores balanceadas, dezenas de acessos ao disco são necessários.
- Somente quase 10 anos mais tarde foram criadas as árvores-B (Bayer, 1971). Um dos motivos da demora foi que a abordagem para criar árvores-B é bastante diferente das outras árvores (por exemplo, árvores-B tem um crescimento *bottom-up*).

Organização de Arquivos

- Árvores-B são a base para implementações comerciais de diversos sistemas (incluindo SGBDs).
- Elas fornecem tempo de acesso proporcional a $log_k N$, com N é o número de entradas e k é o número de entradas em um único bloco.
- Tem termos práticos, árvores-B permitem encontrar uma entrada entre milhões em 3 ou 4 acessos.

Organização de Arquivos

- Outras técnicas como *hashing* externo permitem acessos mais rápidos para arquivos que não sofrem muitas alterações.
- Mas recentemente, *hashing* dinâmico tem sido aplicado para acesso rápido mesmo em grandes arquivos.

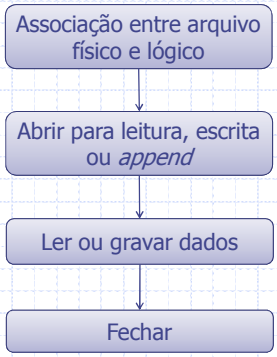
Arquivo Físico e Arquivo Lógico

- **Arquivo Físico:** seqüência de bytes armazenada no disco em blocos distribuídos em trilhas/setores.
- **Arquivo Lógico**: arquivo como visto pelo aplicativo que o acessa – seqüência contínua de registros ou bytes.
- **Associação arquivo físico – arquivo lógico**: iniciada pelo aplicativo, gerenciada pelo S.O.

Arquivo Lógico

- A maioria das linguagens de programação tratam as informações que são lidas e gravadas em arquivos como *streams* de bytes.
- Gerenciar arquivos é portanto gerenciar esses *streams*.
- Algumas operações comuns sobre arquivos são abrir, fechar, ler e gravar além de verificar situações de erro.

Operações sobre Arquivos



Associação entre Arquivo Físico e Arquivo Lógico

- Em Turbo Pascal:

```
file arq;
assign(arq, 'meuarq.dat');
```

- Em C: (associa e abre para leitura)

```
FILE *p_arq;
if ((p_arq=fopen("meuarq.dat", "r"))==NULL)
   printf("Erro na abertura do arquivo\n");
else
```

Abertura de Arquivos

- Arquivo novo (escrita) ou arquivo já existente (leitura ou escrita)
  - Em Turbo Pascal
    - reset( ) – para arquivos existentes
    - rewrite( ) – para arquivos novos
    - assign(arq, "meuarq.dat");
    - reset(arq) ou rewrite(arq);

Abertura de Arquivos

- Em C
  - Comandos

    fopen – comando da linguagem (stdio.h)

    open – comando do sistema (chamada ao sistema operacional UNIX – fcntl.h e file.h)
  - Parâmetros indicam o modo de abertura

Função fopen

- fd=fopen(<filename>,<flags>)
  - filename: nome do arquivo a ser aberto;
  - flags: controla o modo de abertura;
    - "r": Abre para leitura. O arquivo precisa existir;
    - "w": Cria um arquivo vazio para escrita;
    - "a": Adiciona conteúdo ao arquivo (*append*) ;
    - "r+": Abre o arquivo para leitura e escrita;
    - "w+": Cria um arquivo vazio para leitura e escrita;
    - "a+": Abre um arquivo para leitura e adição (*append*);
    - "t": modo texto (*default*) – o fim do arquivo é o primeiro CTRL+Z (DOS) encontrado;
    - "b": modo binário – o fim do arquivo é o último byte;
    - Unix não diferencia arquivos texto e binário.

Função open

- fd=open(<filename>,<flags>,[pmode])
  - fd: descritor (identificador do arquivo lógico). Open retorna NULL em caso de erro;
  - flags: controla o modo de abertura:
    - O_APPEND: abre para escrita no final do arquivo;
    - O_CREAT: cria o arquivo se ele não existe;
    - O_RDONLY: abre apenas para leitura;
    - O_WRONLY: abre apenas para escrita;
    - O_RDWR: abre para leitura e escrita;
    - O_TRUNC: trunca o tamanho do arquivo para zero.
  - pmode: seqüência octal indica permissões de acesso (p/ *owner*, *group*, *world*)
    - Exemplo: pmode=0751 (rwxrw---x)

Fechamento de Arquivos

- Encerra a associação entre arquivos lógico e físico, garantindo que todas as informações sejam atualizadas e salvas (conteúdo dos *buffers* de E/S enviados para o arquivo).
- S.O. fecha o arquivo se o aplicativo não o fizer ao final da execução do programa. Interessante para:
  - Garantir que os *buffers* sejam descarregados;
  - Liberar as estruturas associadas ao arquivo para outros arquivos.

Exemplo: fechamento de arquivos

Pascal: close(arq)

```
assign(arq, 'meuarq.dat');
reset(arq);
...
close(arq);
```

C: close (UNIX) e fclose (C)

```
fd = open("meuarq.dat",O_RDONLY);
...
close(fd);
```

```
fd= fopen("meuarq.dat", "r")
...
fclose(fd)
```

Leitura e Escrita

- C: Operações do S.O. (UNIX)
  - read(<source-file>,<dest-addr>,<size>)
  - write(<destination-file>,<source-addr>,<size>)
    - Retornam o número de bytes lidos/escritos
    - <source-file> e <destination-file>: descritores do arquivo
    - <dest-addr> e <source-addr>: endereço da posição de memória inicial
    - <size>: número de bytes a serem lidos/escritos

Leitura e Escrita

- C: Funções da linguagem
  - Básicas
    - fread(<dest-addr>, <size>, <nr>, <fd>)
    - fwrite(<source-addr>, <size>, <nr>, <fd>)
  - Caracteres
    - fgetc(<fd>)
    - fputc(<caractere>,<fd>)

Leitura e Escrita

- C: Funções da linguagem
  - Formatadas
    - fscanf(<fd>, <format>, ...)
    - fprintf(<fd>, <format>, ...)
  - String
    - fgets(<str>, <size>, <fd>)

Fim de Arquivo

- Ponteiro de arquivo: controla o próximo byte a ser lido.
  - Pascal:
    - eof(arq) (função lógica)
  - C:
    - feof(arq) (função lógica)
    - fread() retorna o número de bytes lidos. Se igual a zero, indica final de arquivo.

Exercício

- Fazer um programa que lê imprime o conteúdo de um arquivo na tela.

Exemplo: Mostrar o conteúdo de um arquivo - Código Pascal

```
var
   arq: file of char;
   filename: string;
   ch: char;

begin
   write('Entre o nome do arquivo: ');
   readln(filename);
   assign(arq, filename);
   reset(arq);
   while (not eof(arq)) do
   begin
      read(arq, ch);
      write(ch);
   end;
   readln;
end.
```

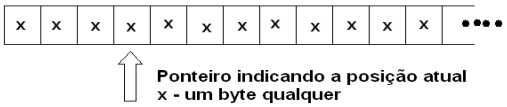
Exemplo: Mostrar o conteúdo de um arquivo - Código C

```
#include<stdio.h>
int main() {
    char ch;
    FILE *file;
    char filename[20];

    printf("Entre o nome do arquivo: ");
    gets(filename);
    file = fopen(filename, "r");
    while (fread(&ch, 1, 1, file) != 0)
          fwrite(&ch, 1, 1, stdout);
    fclose(file);
}
```

(Folk at al., 1998)

O ponteiro no arquivo lógico



Acesso seqüencial X aleatório

- Leitura seqüencial: ponteiro de leitura avança byte a byte (ou por blocos), a partir de uma posição inicial
- Acesso aleatório (direto): posicionamento em um byte ou registro arbitrário

Seeking

- *Seeking*: ação de mover o ponteiro para uma certa posição no arquivo:
  - Unix: seek(<source-file>, <offset>)
    - offset – posição desejada, em bytes, a partir do início do arquivo.
  - C: pos=fseek(<fd>,<byte-offset>,<origin>)
    - Função retorna a posição final do ponteiro;
    - byte-offset – deslocamento, em bytes, a partir de *origin;*
    - Origin: 0 – início do arquivo; 1 – posição corrente; 2 – final do arquivo.

Seeking

- *Seeking*: ação de mover o ponteiro para uma certa posição no arquivo:
  - Pascal: seek(<arq>, <n>)
    - n é o número do registro (iniciando com zero).

Exercícios

1. Escreva um programa que leia da entrada padrão e grave em um arquivo texto.
2. Implemente o comando UNIX `tail -n`, no qual *n* é o número de linhas no final do arquivo a serem copiadas para stdout.