Algoritmos e Estruturas de Dados II
Fundamentos de Arquivos

Professora:
Josiane M. Bueno

Fundamentos de Arquivos
Arquivos

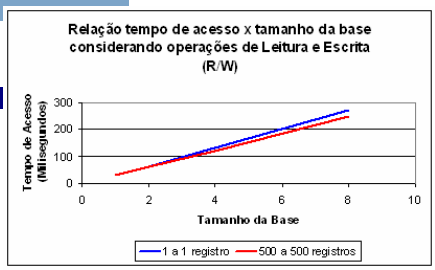
- Informação mantida em memória secundária:
  - HD
  - Disquete
  - Fitas
  - CD
  - DVD
  - Etc.

Fundamentos de Arquivos
Discos X Memória Principal

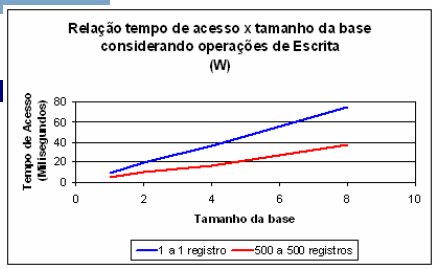
- Tempo de acesso
  - HD: alguns milisegundos ~ 10ms
  - RAM: alguns nanosegundos ~ 10ns...40ns
  - Ordem de grandeza da diferença entre os tempos de acesso ~ 250.000, isto é, HDs são 250.000 vezes mais lentos que memória RAM

Fundamentos de Arquivos
Discos X Memória Principal

- Capacidade de Armazenamento
  - HD – muito alta, a um custo relativamente baixo
  - RAM – limitada pelo custo e espaço
- Tipo de Armazenamento
  - HD – não volátil
  - RAM - volátil



R/W	Tamanho Base (x 5.500 registros)	Tempo Médio em 100 testes (ms) Gravação registro a registro	Gravação 500 a 500 registros
	1	31	29
	2	62	58
	4	131	118
	8	272	249



W	Tamanho Base (x 5.500 registros)	Tempo Médio em 100 testes (ms) Gravação registro a registro	Gravação 500 a 500 registros
	1	9	4
	2	19	10
	4	36	17
	8	75	37

Fundamentos de Arquivos

Discos X Memória Principal

- Em resumo
  - acesso a disco é muito caro, isto é, lento!
- Então
  - o número de acessos ao disco deve ser minimizado
  - a quantidade de informações recuperadas em um acesso deve ser maximizada

Fundamentos de Arquivos

Discos X Memória Principal

- Estruturas de dados eficientes em memória principal são inviáveis em disco
- Seria fácil obter uma estrutura de dados adequada para disco se os arquivos fossem estáveis (não sofressem alterações)
- Como resolver o problema de acesso a dados em disco?

Estruturas adequadas de organização de informação em arquivos

Fundamentos de Arquivos

Organização de Arquivos

- **Meta**: minimizar as desvantagens do uso da memória externa
- **Objetivo** : minimizar o tempo de acesso ao dispositivo de armazenamento externo
- De forma independente da tecnologia:

  Tempo de Acesso =

  no. de acessos * tempo de 1 acesso

Fundamentos de Arquivos

Arquivo Físico e Arquivo Lógico

- **Arquivo Físico**: conjunto bytes no disco, geralmente agrupados em setores de dados. Gerenciado pelo sistema operacional
- **Arquivo Lógico**: modo como a linguagem de programação enxerga os dados. Uma seqüência de bytes, eventualmente organizados em registros ou outra estrutura lógica.
- **Associação arquivo físico com arquivo lógico**: iniciada pelo aplicativo, gerenciada pelo S.O.

Fundamentos de Arquivos

Arquivo Lógico e Arquivo Físico

Associação entre Arquivo Físico e Arquivo Lógico:

- Em Turbo Pascal:

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

- Em C: (associa e abre para leitura)

```
file *parq;
 if ((parq=fopen("meuarq.dat", "r"))==NULL)
    printf("erro...")
else ...
```

Fundamentos de Arquivos

Abertura de Arquivos

- Arquivo novo (p/ escrita) ou arquivo já existente (p/ leitura ou escrita)...
  - Em Turbo Pascal

```
reset( ) //para arquivo existente
rewrite( ) //para criar novo arquivo

Ex:
assign(arq, "meuarq.dat");
reset(arq) ou rewrite(arq);
```

Fundamentos de Arquivos
Abertura de Arquivos

- Em C
  - Comandos

    fopen – comando da linguagem

    open – comando do sistema (chamada ao sistema operacional UNIX)
  - Parâmetros especiais indicam o modo de abertura

Fundamentos de Arquivos
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 – o fim do arquivo é assumido ser o primeiro CTRL+Z encontrado
    - b modo binário – o fim do arquivo é o último byte, seja qual for.

Fundamentos de Arquivos
comando 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 (rwxr-x--x)

Fundamentos de Arquivos
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. Interessante para:
  - Prevenir contra interrupção
  - Liberar as estruturas associadas ao arquivo para outros arquivos.

Fundamentos de Arquivos
Fechamento de Arquivos

- Pascal: close(arq)
- C:

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

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

Fundamentos de Arquivos
Leitura e Escrita

C: Operações do S.O.

- 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

Fundamentos de Arquivos

Leitura e Escrita

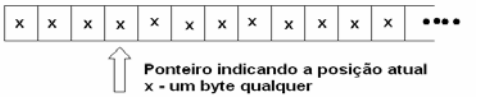
C: Funções da linguagem

- unsigned **fread** (void *buffer, int numero_de_bytes, int count, FILE *fp);
- unsigned **fwrite** (void *buffer,int numero_de_bytes, int count,FILE *fp);
- **ch = fgetc (fd)**
- **fputc (ch, fd)**

Fundamentos de Arquivos

Fim de Arquivo

- Ponteiro de arquivo: controla o próximo byte a ser lido



- Pascal: **eof**(arq) (função lógica)
- C: **read**() retorna o número de bytes lidos. Se igual a zero, indica final de arquivo.

Fundamentos de Arquivos

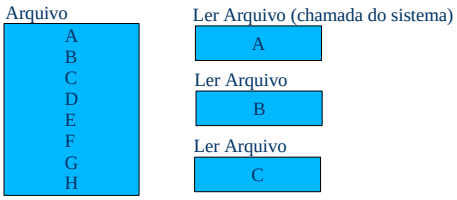
Acesso seqüencial X direto

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

Fundamentos de Arquivos

Acesso seqüencial

A, B, C, ... podem representar bytes, linhas ou registros



Fundamentos de Arquivos

Acesso seqüencial

- Exemplos:
  - Leitura seqüencial de programa fontes
  - Impressão de arquivo
  - Copiar conteúdo de um arquivo para outro
    - Leitura seqüencial do arquivo origem
    - Escrita seqüencial do arquivo destino
- A leitura e escrita seqüencial permitem que o usuário especifique a quantidade de dados a serem lidos ou escritos de cada vez

Fundamentos de Arquivos

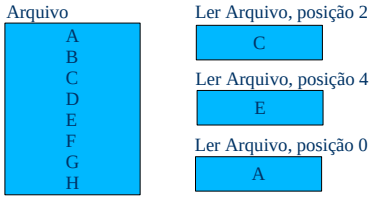
Acesso Direto

Na chamada do sistema, o programa inclui a posição do arquivo a ser lida

- Cada posição do arquivo corresponde a um byte
- Exemplo de registro de funcionário
  - Funcionário 1:
  - Ocupa os bytes de 0 a (TAM-1)
    - Onde TAM é o tamanho de um registro
  - Funcionário 2:
    - De TAM a 2xTAM-1 ...

Fundamentos de Arquivos
Acesso Direto

Adequado, por exemplo, quando se quer acessar os dados da milionésima pessoa em um cadastro



Fundamentos de Arquivos
Acesso Direto

- Em alguns SOs existe o conceito de posição corrente no arquivo.
- Existe uma chamada de sistema "Posicionar"
  - O acesso relativo pode ser obtido por meio desta chamada

Fundamentos de Arquivos
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

- **No Pascal**

  seek(arq,n) – *n* é o número do registro

Fundamentos de Arquivos
Seeking

- **C**

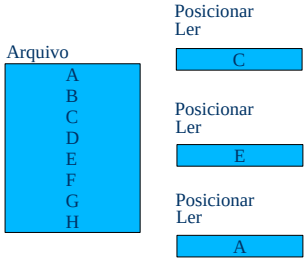
  *pos= fseek (FILE *fp,long byte-offset, int origem);*

  - Função retorna a posição do ponteiro
  - *byte-offset* – deslocamento, em bytes, a partir de *origin*
  - *Origin*

    0 – início do arquivo *1 – posição corrente*
    2 – final do arquivo

Fundamentos de Arquivos
Acesso Direto utilizando Posição Corrente

Seqüencial: caso particular



Fundamentos de Arquivos
Acesso Direto utilizando Posição Corrente

- Utiliza a mesma chamada de sistema para acessos seqüenciais e diretos
- Porém, duas chamadas de sistema são necessárias para realizar o acesso direto ao invés de apenas uma.

Fundamentos de Arquivos

Acesso Direto

O Acesso Direto **não** é suficiente para muitas aplicações:

- É preciso saber a posição adequada do arquivo: sabe-se o nome do registro, mas não seu número.

Fundamentos de Arquivos

Acesso Aleatório

- Solução:
  - **Registros não-ordenados**: Pesquisa seqüencial para encontrar o registro desejado
  - **Registro ordenados**: Pesquisa binária utilizando acesso relativo
- Soluções mais sofisticadas
  - Implementadas por **gerenciadores de bancos de dados**

Fundamentos de Arquivos

Bufferização

- Toda operação de I/O é 'bufferizada'
  - Buffer: I/O de dispositivos exceto discos (teclado, vídeo, etc.)
  - Memória *Cache*: I/O discos 256K, 640K
  - Os bytes passam por uma 'memória de transferência' de tamanho fixo e de acesso otimizado, de maneira a serem transferidos em blocos
  - Porque?

Fundamentos de Arquivos

Bufferização

Qual o tamanho dos blocos de leitura/escrita?

- Depende do SO e da organização do disco

(sistema de arquivo: gerencia a manipulação de dados no disco, determinando como arquivos podem ser gravados, alterados, nomeados ou apagados)

- Ex:

No Windows, é determinado pela FAT (FAT16, FAT32 ou NTFS)

Fundamentos de Arquivos

Bibliografia

- Folk & Zoelick, File Structures;
- Transparências Leandro C. Cintra e M.C.F. de Oliveira.