## Fundamentos de Arquivos

Leandro C. Cintra
M.C.F. de Oliveira
2004
Fonte: Folk & Zoelick, File Structures

## Arquivos

- Informação mantida em memória secundária
  - HD
  - Disquete
  - Fitas
  - CD

## 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

## 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

## 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
- Estruturas de organização de informação em 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

## 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)
  - Solução: Organização adequada de arquivos no disco, e de informações em arquivos

## Arquivo Físico e Arquivo Lógico

- **Arquivo Físico:** seqüência de bytes armazenada no disco
- **Arquivo Lógico**: arquivo como visto pelo aplicativo que o acessa
- **Associação arquivo físico – arquivo lógico**: iniciada pelo aplicativo, gerenciada pelo S.O.

## 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.

## Exemplo: 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 ...
```

## 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

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

## 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

## 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 um CTRL+Z encontrado
    - b modo binário – o fim do arquivo é o último byte, seja qual for.

## 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 (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. Interessante para:
  - Prevenir contra interrupção
  - Liberar as estruturas associadas ao arquivo para outros arquivos.

## Exemplo: fechamento de arquivos

Pascal: close(arq)

C:

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

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

## 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

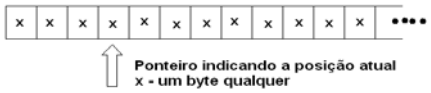
## Leitura e Escrita

- C: Funções da linguagem
  - fread(fd, <dest-addr>)
  - fwrite(fd,<source-addr>
  - fgetc(fd)
  - fputc(fd)

## 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.

## 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=lseek(fd,byte-offset,origin) (fseek)
      - 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
  - No Pascal
    - seek(arq,n) – n é o número do registro

## 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?

## 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)