Fundamentos de Arquivos

Leandro C. Cintra
M.C.F. de Oliveira
Rosane Minghim
2004-2012

PAE 2012: Rafael Martins

Fonte: Folk & Zoelick, File Structure

Arquivos

- Informação mantida em memória secundária
  - HD
  - Fitas magnéticas
  - CD
  - DVD
  - HD – USB

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 de 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;
parq = fopen("meuarq.dat", "r")
```

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 apenas para leitura; o arquivo precisa existir
  - "w" cria arquivo vazio para escrita (se já existe, é apagado)
  - "a" adiciona conteúdo ao arquivo (append)
  - "r+" Abre arquivo para leitura e escrita
  - "w+" Cria arquivo vazio para leitura e escrita
  - "a+" Abre arquivo para leitura e adição (append)
  - t = modo texto – fim do arquivo é o primeiro <CTRL+Z>
  - b = modo binário – fim do arquivo é o último byte

Comando open

```
fd = open(<filename>,<flags>,[pmode])
```

- **fd**: descritor (identificador do arquivo lógico);
  - **open** retorna **NULL** em caso de erro
- **flags**: controla 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**: *(opcional)* 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:

```
// para 'open':
close(fd);
// para 'fopen':
fclose(fd)
```

Leitura e Escrita

- C: Operações do S.O.
- Dados lidos/escritos como bloco de bytes

```
read(<src-file>,<dest-addr>,<size>)
write(<dest-file>,<src-addr>,<size>)
```

- Retornam o número de bytes lidos/escritos
  - <src-file> e <dest-file>: descritores do arquivo
  - <dest-addr> e <src-addr>: endereços de memória
  - <size>: número de bytes a serem lidos/escritos

Leitura e Escrita

- C: Funções da linguagem

```
fread(<dest-addr>, size, count, <src-file>)
fwrite(<src-addr>, size, count, <dest-file>)
```

- Dados lidos/escritos em blocos de bytes (modo binário)

```
<char> = fgetc(<src-file>)
fputc(<dest-file>, <char>)
```

- Dados lidos/escritos um byte (caractere) por vez

Leitura e Escrita

- C: Funções da linguagem

```
fgets(<string>, <max>, <src-file>)
fputs(<string>, <max> <dest-file>)
```

  - dados lidos/escritos como *strings*

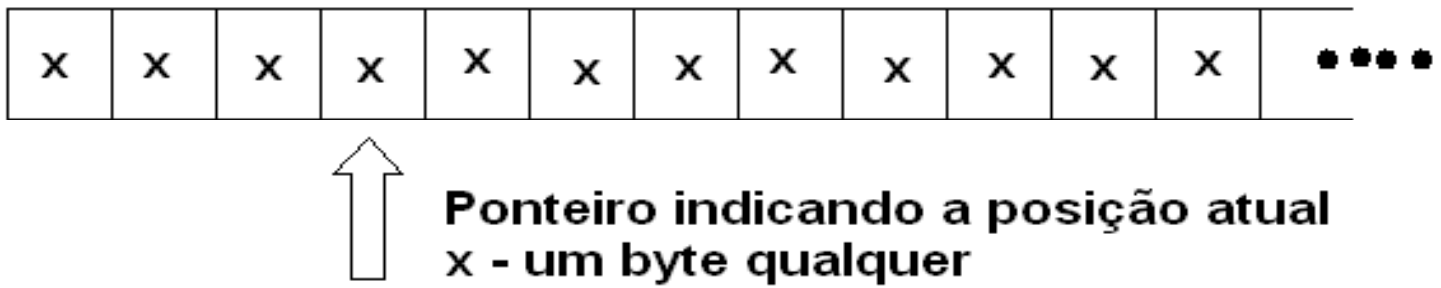
```
fscanf(<src-file>, ...)
fprintf(<dest-file>, ...)
```

  - dados lidos/escritos de modo formatado

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): acesso envolve o posicionamento do ponteiro em um byte ou registro arbitrário

Seeking

- Ação de mover o ponteiro para uma certa posição no arquivo
- Unix: `lseek(<src-file>, <offset>, <origin>)`
- C: `pos=fseek(<src-file>, <offset>, <origin>)`
  - Retorna a posição final do ponteiro
  - *offset:* posição desejada, em bytes, a partir de *origin*
  - *origin:* **`SEEK_SET`** – início do arquivo, **`SEEK_CUR`** – posição atual, **`SEEK_END`** – fim do arquivo
- 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)

FIM