COS110 – Algoritmos e Programação – Módulo 10

Priscila Machado Vieira Lima

Junho/2018

Curso : Engenharia de Controle e Automação

NCE

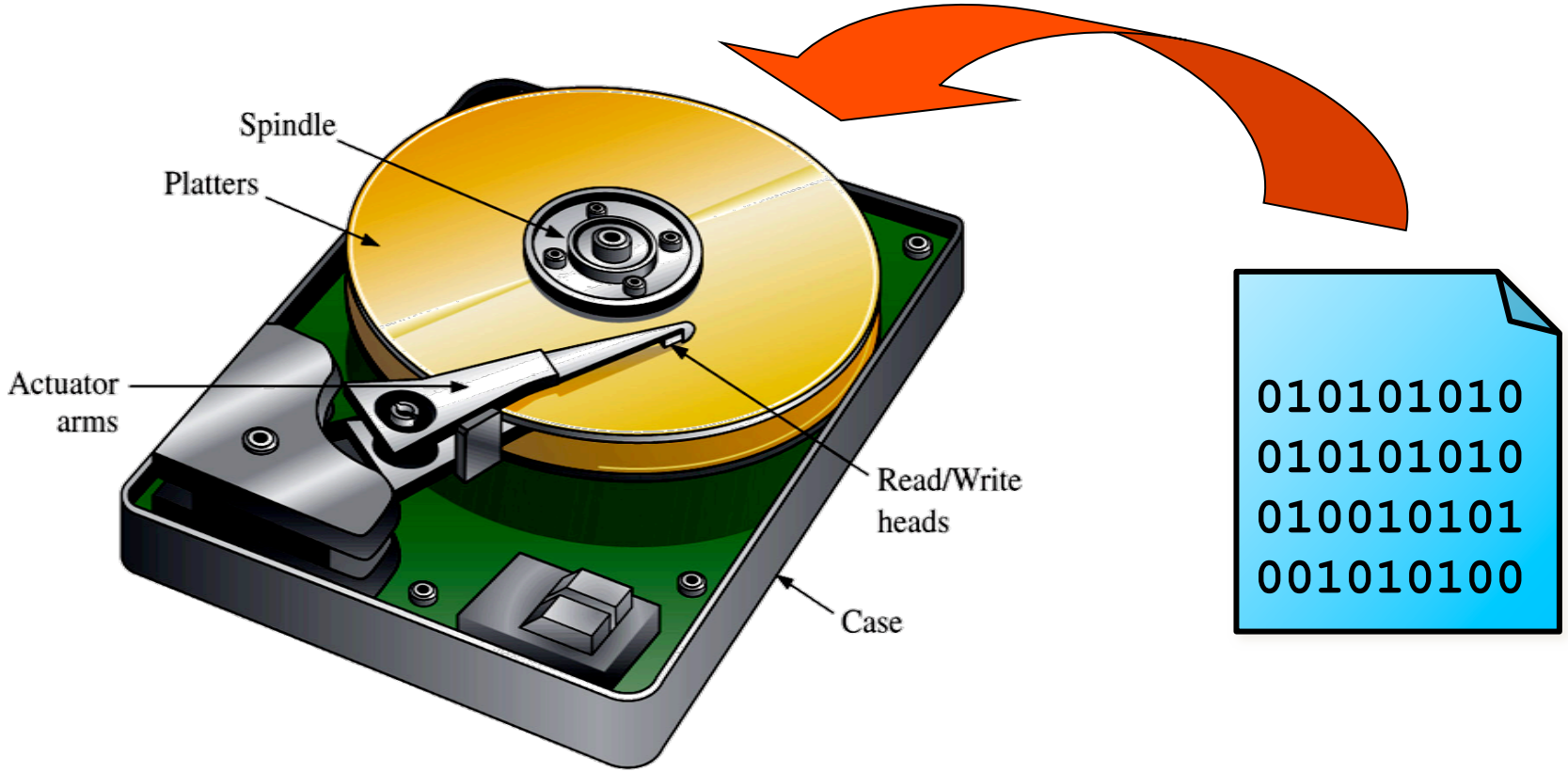
Universidade Federal do Rio de Janeiro

Roteiro

- **Arquivos**

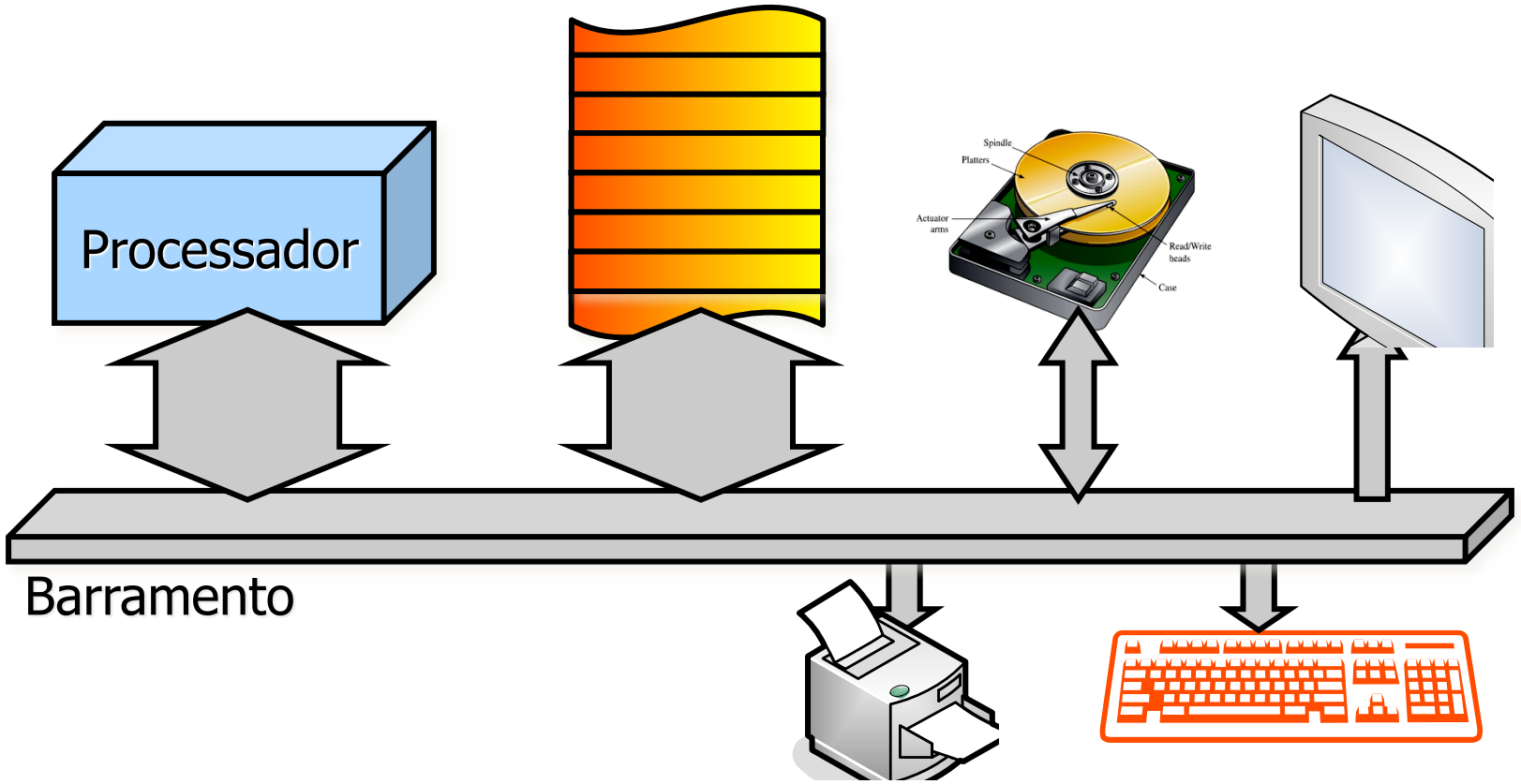
Conceitos iniciais

- Um arquivo (file) é uma seqüência de bytes que reside em uma área de armazenamento (Ex: disco magnético, flash drive, CD-ROM).



Conceitos iniciais

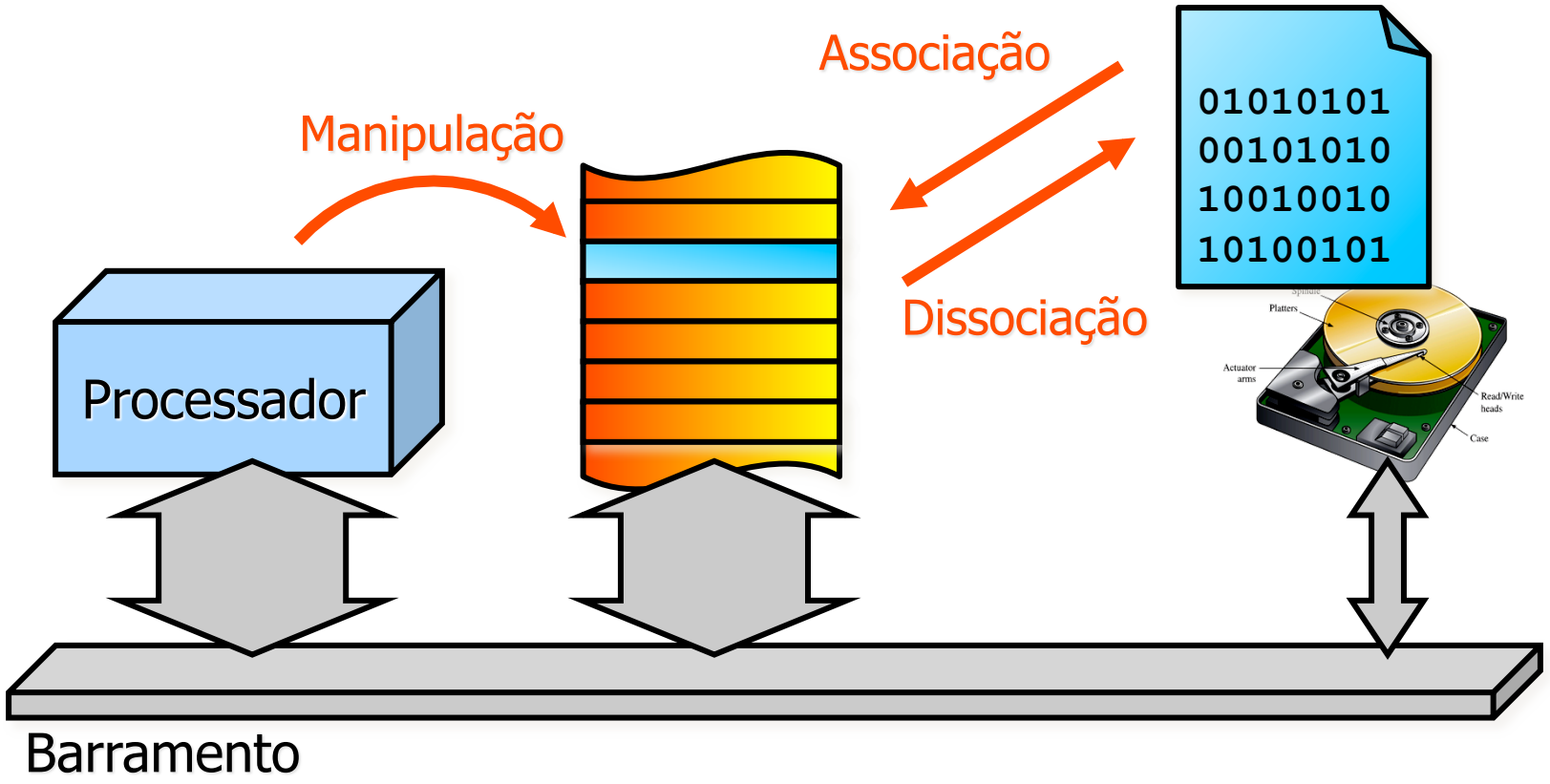
- A Linguagem C acessa um arquivo ou qualquer outro dispositivo de entrada e saída através de um endereço da memória principal.



Conceitos iniciais

- Portanto, arquivos não são acessados diretamente pelo processador quando ele executa um programa.
- Para manipular arquivos, é preciso associá-los a um stream e, então, manipular o stream.
- A associação de um arquivo a um stream é realizada através de uma operação de abertura.
- A dissociação é realizada por meio de uma operação de fechamento.

Conceitos iniciais



Driver e *Drive*

- *Drive* –componente físico sua máquina que serve como unidade de armazenamento
  - Exs: *drives* de CD, DVD e Blu-ray,.
- *Driver* – atua como controlador (“motorista”), transmitindo e interpretando dados entre o sistema operacional e uma peça de *hardware*.
  - Ex: placa de vídeo *off-board* para incrementar a capacidade de processamento gráfico de um computador. tendo suas próprias diretrizes e recursos que precisam ser identificados pelo sistema operacional. Uma nova impressora também precisa ter seu *driver* instalado.
  - *drivers* precisam ser instalados (os CDs ou pendrives de instalação costumam acompanhar o produto ou podem ser baixados do site correspondente) e, eventualmente, atualizados..

Características dos arquivos

- Podem armazenar grande quantidade de informação.
- Dados são persistentes (gravados em disco).
- Acesso aos dados pode não ser seqüencial (acesso direto a registro de um banco de dados).
- Acesso à informação pode ser concorrente (mais de um programa ao mesmo tempo).

Nomes e extensões

- Arquivos são identificados por um nome.

- O nome de um arquivo pode ter uma extensão que indica o tipo do conteúdo do arquivo.

```
arquivo.ext
```

Tipos de arquivos

- Arquivo texto: Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por editores de texto simples.
  - Exemplos: código C, texto simples, páginas HTML.

- Arquivo binário: Seqüência de bits sujeita às convenções dos programas que o gerou, não legíveis diretamente.
  - Exemplos: arquivos executáveis ou compactados, documentos do Word.

Caminhos absolutos ou relativos

- O nome de um arquivo pode conter o seu diretório, ou seja, o caminho (*path*) para encontrar tal arquivo.
- Os caminhos podem ser especificados de duas formas:
  - Caminho absoluto: descrição desde o diretório raiz

```
/bin/emacs
/home/usr1/arq.txt
/Users/priscila21/Desktop/Espacco-CodeBlocks/TestaVetStruct
/Users/priscila21/Desktop/Espacco-CodeBlocks/TestaVetStruct.c
```

  - Caminho relativo: descrição desde o diretório corrente.

```
arq.txt
mc102/lab.c
TestaVetStruct.c
Desktop/Espacco-CodeBlocks/TestaVetStruct
```

Ponteiro de arquivo

- Como já comentado, a associação de um arquivo a um stream é realizada pela operação de abertura.

- A abertura de um arquivo retorna um ponteiro especial para o início do arquivo, conhecido como ponteiro de arquivo.

- Basicamente, o ponteiro de arquivo identifica um arquivo específico em disco e é utilizado pelo stream associado para direcionar as operações de entrada/saída (E/S).

Ponteiro de arquivo

- Um ponteiro de arquivo deve ser declarado como sendo do tipo FILE.

```
FILE *arq;
```

- O tipo FILE está definido na biblioteca **stdio.h**
- As funções que manipulam um arquivo (ex.: escrita, leitura) devem ser realizadas sobre o ponteiro declarado.

Funções para manipulação de arquivos

Função	Finalidade
`fopen()`	Abre um arquivo
`fclose()`	Fecha um arquivo
`fputc()`	Escreve um caractere em um arquivo
`fgetc()`	Lê um caractere de um arquivo
`fputs()`	Escreve uma string em um arquivo
`fgets()`	Lê uma string de um arquivo
`fprintf()`	É para um arquivo o que printf() é para o console

Funções para manipulação de arquivos

Função	Finalidade
`fscanf()`	É para um arquivo o que scanf() é para o console
`fwrite()`	Escreve tipos de dados maiores que um byte em arquivo
`fread()`	Lê tipos de dados maiores que um byte em arquivo
`feof()`	Devolve verdadeiro se o fim de arquivo for atingido
`ferror()`	Devolve verdadeiro se ocorreu um erro
`remove()`	Apaga um arquivo
`fseek()`	Posiciona o arquivo em um byte específico

Abrindo um arquivo
:: Função `fopen()`

- Abre um arquivo para leitura e/ou escrita.
- Retorna um ponteiro para o arquivo.
- Nunca se deve alterar o valor desse ponteiro.
- É a única função que tem o nome do arquivo como argumento. Todas as demais funções utilizam o valor do ponteiro para indicar que arquivo estão manipulando.
- O parâmetro **<modo>** determina como o arquivo será aberto.

```
fopen(<nome do arquivo>, <modo>)
```

Abrindo um arquivo :: Modo

Modo	Significado
r	Abre um arquivo texto para leitura
w	Cria/sobrescreve um arquivo texto para escrita
a	Anexa a um arquivo texto existente
rb	Abre um arquivo binário para leitura
wb	Cria/sobrescreve um arquivo binário para escrita
ab	Anexa a um arquivo binário existente
r+	Abre um arquivo texto para leitura e escrita
w+	Cria/sobrescreve um arquivo texto para leitura e escrita
rb+	Abre um arquivo binário para leitura e escrita
wb+	Cria/sobrescreve um arquivo binário para leitura e escrita

Abrindo um arquivo
:: Função `fopen()`

- Caso ocorra um erro na abertura do arquivo, esta função retornará um ponteiro vazio (NULL).

- Deve-se sempre testar o sucesso de **`fopen()`** antes de tentar qualquer outra operação sobre o arquivo.

- O número máximo de arquivos que podem estar abertos simultaneamente é dado pela macro FOPEN_MAX, da biblioteca **`stdio.h`**.

Fechando um arquivo
:: Função `fclose()`

- Serve para dissociar uma stream de um arquivo aberto pela função **`fopen()`**.
- Em caso de sucesso, **`fclose()`** retorna 0 (zero). Qualquer outro valor indica erro no fechamento do arquivo indicado.

```
fclose(<pt_arquivo>);
```

Escrevendo um caractere :: Função **fputc()**

- O padrão C ANSI define duas funções equivalentes para escrever caracteres em um arquivo: **putc()** e **fputc()**.
- Ambas escrevem caracteres em um arquivo que foi previamente aberto por **fopen()**.

```
putc(<caractere>, <pt_arquivo>);
fputc(<caractere>, <pt_arquivo>);
```

Lendo um caractere
:: Função `fgetc()`

- Para ler um caractere em um arquivo aberto por fopen(), pode-se usar as funções getc() ou fgetc().

```
var = fgetc(<pt_arquivo>);
```

- A função devolve EOF quando o final do arquivo é alcançado.

Lendo um caractere
:: Função `getc()`

- No código abaixo, o arquivo é lido até que a marca de final de arquivo (EOF – *End of File*) seja alcançada.

```
do {
   ch = fgetc(pt_arq);
} while (ch != EOF);
```

Lendo uma string
:: Função `fgets()`

- Lê uma string de caracteres da stream especificada até que um caractere de nova linha seja lido ou que length-1 caracteres sejam lidos.
- Se lido, o caractere de nova linha (\n) faz parte da string.
- A string resultante é terminada por um caractere nulo (\0).

```
fgets(string, lenght, pt_arquivo);
```

Escrevendo uma string
:: Função `fputs()`

- Grava string de caracteres na stream especificada.
- Devolve EOF se ocorrer erro.

```
fputs(string, pt_arquivo);
```

Lendo e escrevendo estruturas de dados :: Funções **`fread()`** e **`fwrite()`**

- Permitem a leitura e escrita de blocos de qualquer tipo de dado.

```
fread(buffer, no_bytes, no_itens, pt_arq);
fwrite(buffer, no_bytes, no_itens, pt_arq);
```

- **`buffer`** é um ponteiro para memória que receberá/ fornecerá os dados lidos/escritos no arquivo.
- **`no_bytes`** é o número de bytes a ler/escrever.
- **`no_itens`** determina quantos itens serão lidos/ escritos, cada um de comprimento **`no_bytes`**.

Lendo e escrevendo estruturas de dados :: Funções `fread()` e `fwrite()`

- A função **`fread()`** devolve o número de itens lido e a função **`fwrite()`** devolve o número de itens escritos.

- Se tais valores forem menores que o campo **`no_itens`**, é porque o final do arquivo (EOF) foi atingido, ou ocorreu um erro.

- Uma das aplicações mais úteis dessas funções envolve ler e escrever tipos de dados definidos pelo usuário, especialmente estruturas.

Lendo e escrevendo dados formatados :: Funções **`fscanf()`** e **`fprintf()`**

- Essas funções funcionam exatamente como **`printf()`** e **`scanf()`**, exceto por operarem com arquivos.

```
fprintf(pt_arq, string_controle, argumentos);
fscanf(pt_arq, string_controle, argumentos);
```

- Note que **`fprintf()`** e **`fscanf()`** direcionam suas operações de entrada e saída formatadas para o arquivo apontado por **`pt_arq`**.

Lendo e escrevendo dados formatados :: Funções `fscanf()` e `fprintf()`

- Embora essas duas funções sejam a maneira mais fácil de escrever e ler dados em arquivos de disco, nem sempre são a escolha mais apropriada.

- Como os dados são escritos em ASCII e formatados como apareceriam na tela (e não em binário), um tempo extra é perdido a cada chamada.

- Portanto, se há preocupação com velocidade ou tamanho de arquivo, deve-se utilizar as funções **`fread()`** e **`fwrite()`**.

Biblioteca `stdio.h`

- O arquivo de cabeçalho **`stdio.h`** também define várias macros como: NULL, EOF, FOPEN_MAX, SEEK_SET, SEEK_CUR e SEEK_END.
  - A macro NULL define um ponteiro nulo.
  - A macro EOF corresponde ao valor inteiro -1, e indica o final de um arquivo.
  - As outras macros são utilizadas pela função fseek().

Acesso aleatório :: Função **`fseek()`**

- Operações de leitura e escrita aleatórias podem ser realizadas com a ajuda da função **`fseek()`** , que modifica o indicador de posição de arquivo.

```
fseek(pt_arq, no_bytes, origem);
```

- **`no_bytes`** é o número de bytes, a partir de **`origem`**, que se deseja avançar.
- **`origem`** é uma das seguintes macros:

Acesso aleatório :: Função `fseek()`

Origem	Macro
Início do arquivo	SEEK_SET
Posição atual	SEEK_CUR
Final do arquivo	SEEK_END

- A função **`fseek()`** pode ser utilizada para efetuar movimentações em múltiplos de qualquer tipo de dado, simplesmente utilizando-se o comando **`sizeof()`**.