INE5412 – Sistemas Operacionais I

Sistemas de Arquivos

Fundamentos e chamadas de sistema

Prof. Márcio Castro

marcio.castro@ufsc.br

Na aula de hoje

- Introdução
- Arquivos
- Diretórios
- Chamadas de sistema para manipulação de arquivos
- Implementação do sistema de arquivos
  - Arquivos
  - Diretórios

Introdução

- Aplicações precisam **armazenar** e **recuperar** informação
  - A capacidade de armazenamento está restrita ao tamanho do espaço de endereçamento virtual
  - Para alguma aplicações, o tamanho é adequado; para outras, não...
  - **Exemplos:** sistemas de reservas de passagens aéreas e bancos

Introdução

- Em muitas aplicações a informação precisa ficar **retida quando o processo termina**
  - **Exemplo:** bancos de dados
  - Nesses sistemas, é inaceitável que a informação em uso pelo processo desapareça quando ele é encerrado

Introdução

- **Questões a serem gerenciadas pelo S.O.**
  - Como encontrar uma informação?
  - Como impedir que um usuário tenha acesso a informações de outro usuário?
  - Como saber quais blocos estão livres?

Introdução

- **Anteriormente vimos que:**
  - O S.O. abstrai o conceito de **processador** para criar a **abstração de um processo**
  - O S.O. abstrai o conceito de **memória física** para oferecer um **espaço de endereçamento virtual**
- **Armazenamento de dados**
  - Abstrai o conceito de disco, criando a abstração de **arquivos**

ARQUIVOS

Arquivos

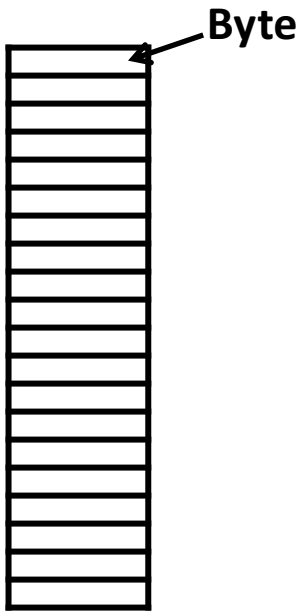
- **Conceito de arquivo**
  - Um arquivo é uma **unidade lógica de informação** que pode ser **criada**, **alterada** e **removida** por um processo
- **Persistência**
  - A informação armazenada em arquivos deve ser **persistente**
  - Não pode ser afetada pela criação ou pelo término de um processo

Arquivos

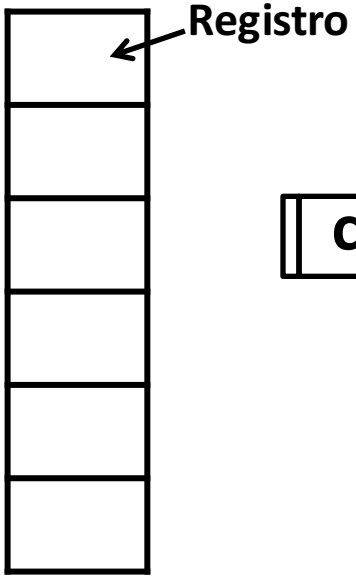
- Arquivos são gerenciados pelo S.O.
- O S.O. define como os arquivos são:
  - Nomeados
  - Estruturados
  - Acessados
  - Protegidos
  - Implementados
- **Sistema de arquivos**: parte do S.O. que trata dos arquivos

Estrutura de arquivos

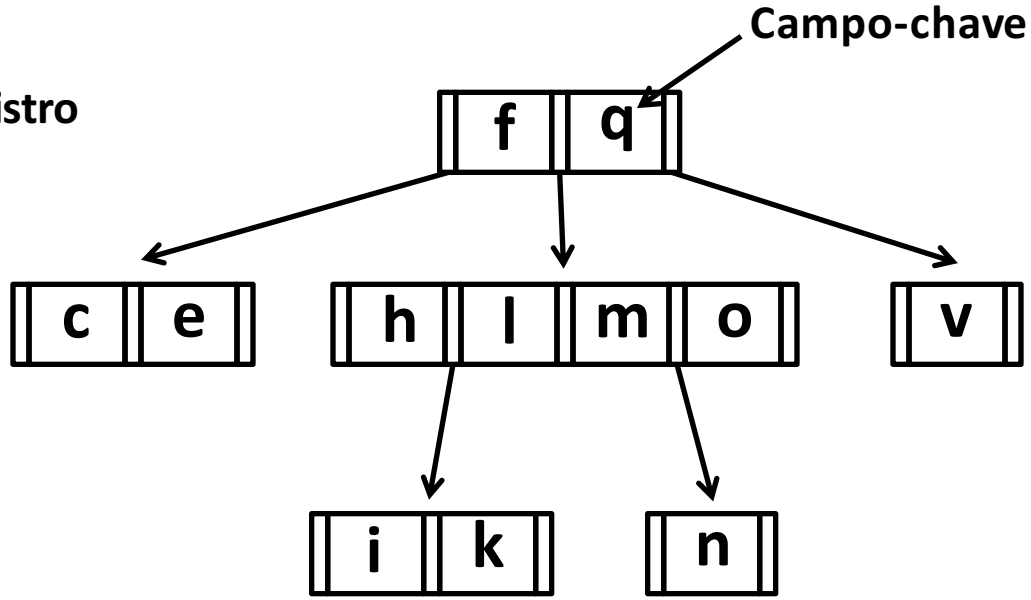
- **Estrutura de arquivos**
  - Os arquivos podem ser estruturados de várias maneiras
  - Alguns exemplos:



(a) sequência desestruturada de bytes



(b) sequência de registros de tamanho fixo



(c) árvore de registros contendo um campo chave em uma posição fixa no registro (árvore ordenada pela chave)

Tipos de arquivos

- **Tipos de arquivos**
  - Muitos S.Os. dão suporte a vários tipos de arquivos
  - **Arquivos regulares:** são os arquivos de usuário
  - **Diretórios:** são arquivos do sistema que mantém a estrutura do sistema de arquivos

Tipos de arquivos

- Arquivos regulares podem ser de dois tipos
- **Arquivos de texto**
  - São constituídos de linhas de texto
  - Cada linha termina com um caractere de retorno de carro (*carriage return*) ou um caractere de próxima linha (*line feed*)
  - São mostrados e impressos diretamente, podendo ser editados com qualquer editor de texto
- **Arquivos binários:**
  - Possuem uma estrutura interna conhecida pelos programas que os manipulam
    - Exemplo: *Executable and Linkable Format (ELF)*

Acesso aos arquivos

- **Acesso aos arquivos**
  - Define a forma com que os bytes de um arquivo podem ser lidos
- **Arquivos de acesso sequencial:**
  - Os bytes são lidos **sequencialmente** partindo-se do início (não permite saltos ou acessos fora de ordem)
  - Convenientes quando o meio de armazenamento era fita magnética
- **Arquivos de acesso aleatório:**
  - Os bytes podem ser lidos **em qualquer ordem**
  - Convenientes para o meio de armazenamento em discos rígidos

Atributos de arquivos

- **Atributos de arquivos**
  - Todo arquivo possui um nome e seus dados
  - Além disso, o S.O. associa outras informações a cada arquivo
  - Essas informações extras são chamadas de **atributos do arquivo**
  - A lista de atributos de um arquivo varia de um S.O. para outro

Atributos de arquivos

Atributo	Significado
Proteção	Quem tem acesso ao arquivo e de que modo
Senha	Necessidade de senha para acesso ao arquivo
Criador	ID do criador do arquivo
Proprietário	Proprietário atual
Flag de somente leitura	0 para leitura/escrita; 1 para somente leitura
Flag de oculto	0 para normal; 1 para não exibir o arquivo
Flag de sistema	0 para arquivos normais; 1 para arquivos do sistema
Flag de arquivamento	0 para arquivos com backup; 1 para arquivos sem backup
Flag de ASCII/binário	0 para arquivos ASCII; 1 para arquivos binários
Flag de acesso aleatório	0 para acesso somente sequencial; 1 para acesso aleatório
Flag de temporário	0 para normal; 1 para apagar o arquivo ao sair do processo
Flag de travamento	0 para destravados; diferente de 0 para travados
Tamanho do registro	Número de bytes em um registro
Posição da chave	Posição da chave em cada registro
Tamanho do campo-chave	Número de bytes no campo-chave
Momento de criação	Data e hora de criação do arquivo
Momento do último acesso	Data e hora do último acesso do arquivo
Momento da última alteração	Data e hora da última modificação do arquivo
Tamanho atual	Número de bytes no arquivo
Tamanho máximo	Número máximo de bytes no arquivo

Operações com arquivos

- **Operações com arquivos**
  - S.Os. diferentes oferecem meios diferentes para manipular arquivos
  - Normalmente a manipulação de arquivos é feita estritamente através **de chamadas de sistema**
  - Isso garante, por exemplo, que um determinado usuário não possa modificar um arquivo de outro usuário (proteção)

Operações com arquivos

- **Create**
  - Cria um arquivo vazio (sem dados)
  - Define alguns atributos básicos do arquivo
- **Delete**
  - Remove o arquivo do disco, liberando o espaço utilizado por ele

Operações com arquivos

- **Open**
  - Antes de usar um arquivo, o processo necessita abri-lo
  - Permite que o S.O. busque e coloque na RAM os atributos e a lista de endereços do disco
- **Close**
  - Fecha o arquivo liberando espaço na RAM
  - Força a escrita do último bloco do arquivo, mesmo que o bloco ainda não esteja completo

Operações com arquivos

- **Seek**
  - Reposiciona o ponteiro de arquivo para um local específico do arquivo
- **Read**
  - Le um conjunto de bytes do arquivo
  - Normalmente os bytes são lidos da posição atual
  - Deve ser especificado na chamada de sistema a quantidade de bytes a serem lidos e um buffer para armazenar os dados lidos
- **Write**
  - Escreve um conjunto de bytes no arquivo
  - Normalmente os bytes são escritos na posição atual
  - Se a posição atual for no final do arquivo, o arquivo do arquivo sofre um aumento; caso contrário, os dados existentes são sobrescritos

Operações com arquivos

- **Append**
  - É uma forma de escrita (write)
  - Escreve os bytes sempre no final do arquivo
- **Get attributes**
  - Retorna os atributos do arquivo
- **Set attributes**
  - Modifica os atributos do arquivo
- **Rename**
  - Altera o nome do arquivo

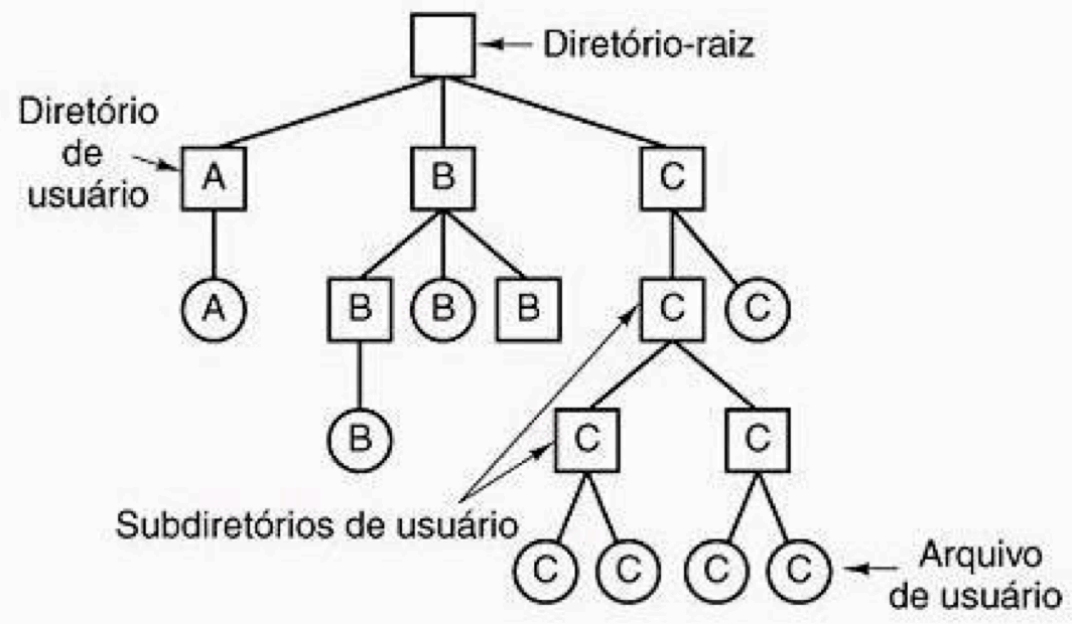
DIRETÓRIOS

Diretórios

- Para organizar os arquivos, muitos S.Os. implementam o conceito de **diretório**
  - Em grande parte deles, um diretório nada mais é do que **um arquivo**!
- **Sistemas de diretórios hierárquicos**
  - Permitem organizar os diretórios de maneira hierárquica
  - Cada usuário pode ter sua própria hierarquia de diretórios

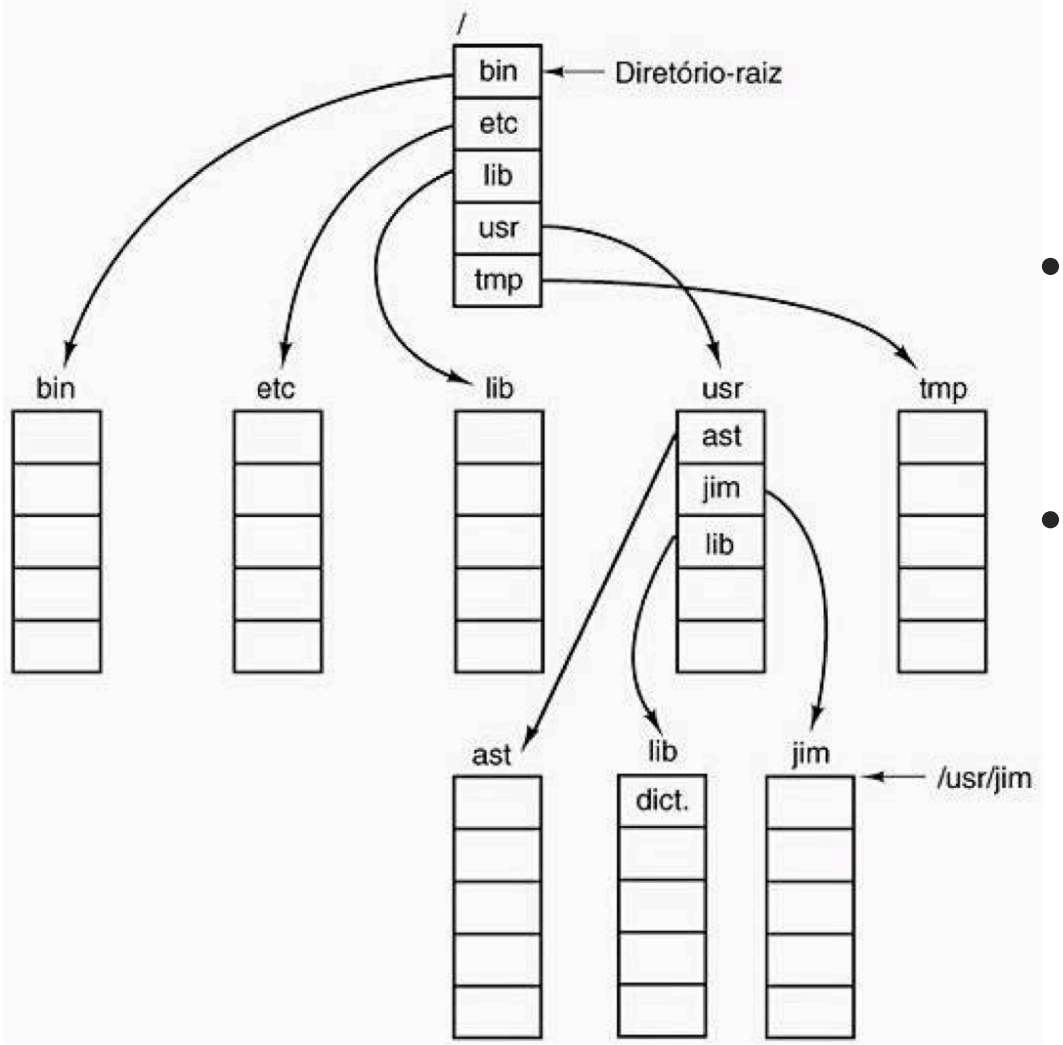
Diretórios hierárquicos

Exemplo de um sistema hierárquico de diretórios



Diretórios hierárquicos

Exemplo de uma árvore de diretórios UNIX



- **Duas entras especiais em cada diretório**
  - '.': diretório atual
  - '..': diretório pai
- **Caminho absoluto**
  - Sempre a partir da raiz
  - */usr/lib/dict*
- **Caminho relativo**
  - A partir do **diretório de trabalho (diretório atual)**
  - **Exemplo:** se o diretório de trabalho é */usr/ast*, um caminho relativo para dict é: *../lib/dict*

Operações com diretórios

- **Operações com diretórios**
  - S.Os. diferentes oferecem meios diferentes para manipular diretórios
  - Assim como em arquivos, a manipulação de diretórios é feita estritamente através de **chamadas de sistema**
  - Isso garante, por exemplo, que um determinado usuário não possa modificar um diretório de outro usuário (**proteção**)

Operações com diretórios

- **Create**
  - Cria um diretório vazio
  - Adiciona automaticamente as entradas especiais ‘.’ e ’..’
- **Delete**
  - Remove um diretório
  - Normalmente, somente diretórios vazios podem ser removidos
  - Diretórios com somente ‘.’ e ‘..’ são considerados vazios

Operações com diretórios

- **Opendir**
  - Permite a leitura de um diretório
  - **Exemplo:** para listar todos os arquivos de um diretório, um programa de listagem abre o diretório para ler os nomes de todos os arquivos que ele contém
- **Closedir**
  - Quando acabar de ser lido, o diretório deve ser fechado para liberar espaço na memória

Operações com diretórios

- **Readdir**
  - Retorna a próxima entrada em um diretório aberto
- **Rename**
  - Modifica o nome de um diretório
- **Link**
  - Cria uma ligação simbólica entre um diretório e um arquivo, permitindo que um mesmo arquivo “apareça” em mais de um diretório
- **Unlink**
  - Remove uma entrada do diretório
  - Se o arquivo estiver presente em apenas um diretório, então o arquivo é removido; caso contrário, somente a ligação simbólica nesse diretório será desfeita

CHAMADAS DE SISTEMA PARA MANIPULAÇÃO DE ARQUIVOS

Chamadas de sistema

- O sistema de E/S de C utiliza o conceito de **streams** e **arquivos**
- Uma stream é um **dispositivo lógico** que representa um **arquivo** ou **dispositivo**
- A stream é **independente do arquivo ou dispositivo**
  - Funções que a manipulam podem escrever tanto em um arquivo no disco quanto em algum outro dispositivo, como o monitor

Chamadas de sistema

- Um **arquivo** é interpretado pela linguagem C como **qualquer dispositivo**, desde um **arquivo em disco** até um **terminal** ou uma **impressora**
  - Exemplo: stdin (teclado), stdout (terminal)

- Para utilizar um arquivo é necessário associá-lo a uma stream e, então, manipular a stream

Chamadas de sistema

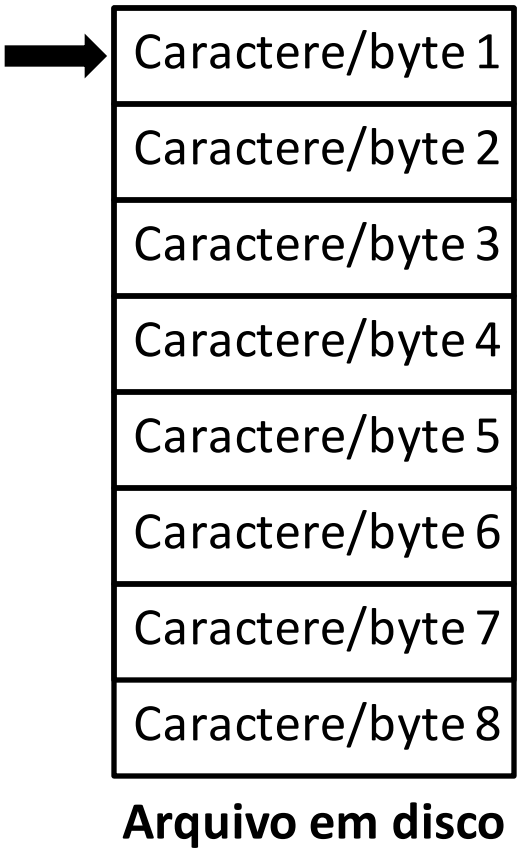
- Existem 2 tipos de streams
  - Texto
  - Binária
- **Stream de texto**
  - Conjunto de **caracteres**
  - **Arquivo texto**
- **Stream binária**
  - Sequência de **bytes**
  - **Arquivo binário**

Chamadas de sistema

- Nem todos os arquivos tem os mesmos recursos
  - Arquivo em disco permite acesso aleatório
  - Um teclado não permite acesso aleatório
- **Logo**
  - Streams são iguais
  - Arquivos associados a streams são diferentes

Chamadas de sistema

- Arquivos em disco
  - Abrir um arquivo em modo leitura ou escrita, faz com que o **indicador de posição** aponte para o começo do arquivo
  - Quando cada caractere ou byte é lido ou escrito no arquivo, o **indicador de posição é incrementado**



Arquivo em disco

Chamadas de sistema

- Associar/desassociar uma stream de um arquivo
  - Associação feita através de uma operação de abertura (**open**)
  - Um arquivo é desassociado de uma stream através de uma operação de fechamento (**close**)
- A operação "close" garante que o conteúdo seja escrito no dispositivo externo automaticamente
  - Esse processo é geralmente chamado de descarga (**flushing**) da stream e garante que nenhuma informação seja acidentalmente deixada no buffer de disco

OPERAÇÕES BÁSICAS EM ARQUIVOS

Operações básicas em arquivos

- **Ponteiro de arquivo**
  - Um ponteiro de arquivo **identifica um arquivo específico** e é usado pela stream para direcionar as operações das funções de E/S
  - Um ponteiro de arquivo é uma variável **ponteiro do tipo FILE**
  - Para ler ou escrever em arquivos é necessário usar o **ponteiro de arquivo**

Exemplo

```
FILE *arquivo;
```

Operações básicas em arquivos

- **Abrindo um arquivo**
  - **fopen(nome_arquivo, modo):** abre uma stream e a associa a um arquivo
  - **nome_arquivo:** *string* contendo o caminho absoluto ou relativo para o arquivo (ex.: "/home/user1/arquivo.txt")
  - **modo:** *string* que representa como o arquivo será aberto
  - **Retorno:** fopen devolve um ponteiro de arquivo ou NULL caso ocorra um problema

Exemplo

```
FILE *arquivo;
if((arquivo = fopen("teste", "r")) == NULL) {
   printf("Erro ao abrir arquivo.\n");
   return(1);
 }
```

Abrindo um arquivo

Modo	Tipo	Função
"r"	Texto	Leitura. Arquivo deve existir.
"w"	Texto	Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"a"	Texto	Escrita. Os dados serão adicionados no fim do arquivo ("append").
"rb"	Binário	Leitura. Arquivo deve existir.
"wb"	Binário	Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"ab"	Binário	Escrita. Os dados serão adicionados no fim do arquivo ("append").
"r+"	Texto	Leitura/Escrita. O arquivo deve existir e pode ser modificado.
"w+"	Texto	Leitura/Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"a+"	Texto	Leitura/Escrita. Os dados serão adicionados no fim do arquivo ("append").
"r+b"	Binário	Leitura/Escrita. O arquivo deve existir e pode ser modificado.
"w+b"	Binário	Leitura/Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"a+b"	Binário	Leitura/Escrita. Os dados serão adicionados no fim do arquivo ("append").

Operações básicas em arquivos

- **Fechando um arquivo**
  - **fclose(arquivo):** fecha uma stream associada a um arquivo
  - **arquivo:** ponteiro de arquivo (FILE *)
  - Escreve qualquer dado restante no buffer de disco no arquivo e o fecha em nível de sistema operacional

Exemplo

```
fclose(arquivo);
```

Funções auxiliares (1)

- Apontando para o início do arquivo
  - **rewind(arquivo)**
- Apontando para uma posição específica dentro do arquivo
  - **fseek(arquivo, offset, origem)**
  - **offset:** aceita valores inteiros positivos ou negativos
  - **origem:**
    - SEEK_SET: a partir do inicio
    - SEEK_CUR: a partir da posição atual do indicador de posição
    - SEEK_END: a partir do final do arquivo
  - Usado em conjunto com sizeof()

Exemplos

```
fseek(arquivo, 10*sizeof(char), SEEK_SET);
fseek(arquivo, -20*sizeof(char), SEEK_END);
```

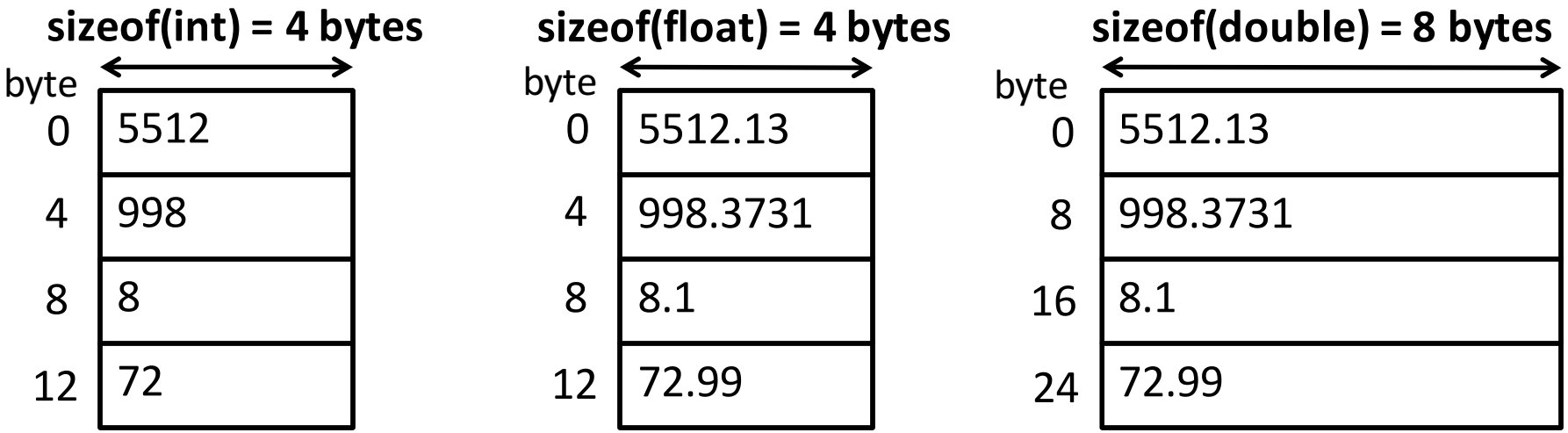
Funções auxiliares (2)

- Removendo um arquivo
  - **remove(nome_arquivo)**
  - **nome_arquivo:** *string* contendo o nome do arquivo
- Renomeando/movendo um arquivo
  - **rename(nome_arquivo_antigo, nome_arquivo_novo)**
- Esvaziando o conteúdo de uma stream aberta para saída
  - **fflush(arquivo)**
  - **arquivo:** ponteiro de arquivo (FILE *)

MANIPULANDO ARQUIVOS BINÁRIOS

Arquivos binários

- Arquivos binários permitem leitura/escrita de conjuntos de bytes
- Esses bytes podem ser estruturados de diferentes formas
  - **Exemplo 1:** bytes representam dados de algum tipo básico da linguagem
    - chars, ints, floats, doubles, ...



Arquivos binários

- Arquivos binários permitem leitura/escrita de conjuntos de bytes
- Esses bytes podem ser estruturados de diferentes formas
  - **Exemplo 2:** bytes representam dados organizados em alguma estrutura definida pelo usuário (**struct**)

```
typedef struct {
    int a;
    float b;
    char c[3];
} minha_estrutura;
```

sizeof(minha_estrutura) = 11 bytes

byte	a	b	c
0	843	9973.365	“gf\0”
11	123	331.9983	“ab\0”
22	5	7.2	“kv\0”
33	-78	-836.83	“oi\0”

Arquivos binários

- Escrevendo e lendo bytes
  - **fwrite(buffer, tamanho, quantidade, arquivo):** escreve um certo número de itens armazenados em um buffer **na posição apontada pelo indicador de posição**
  - **fread(buffer, tamanho, quantidade, arquivo):** lê um certo número de itens começando **na posição apontada pelo indicador de posição** armazenando-os no buffer
  - **Parâmetros**
    - **buffer:** variável de um tipo básico (ou definido pelo usuário), ou um arranjo de um tipo básico (ou definido pelo usuário)
    - **tamanho:** tamanho em bytes de um item (usar sizeof())
    - **quantidade:** quantidade de itens a serem lidos/escritos
    - **arquivo:** é um ponteiro de arquivo (FILE *)
  - Ambas operações incrementam o indicador de posição automaticamente

Exemplo

```
int numero = 123;
fwrite(&numero, sizeof(int), 1, arquivo);
```