Sistemas Operacionais

Gestão de arquivos - uso de arquivos

Prof. Carlos Maziero

DInf UFPR, Curitiba PR

Maio de 2019

Conteúdo

Interface de acesso

Representação lógica do arquivo: *descritor* ou *handle*

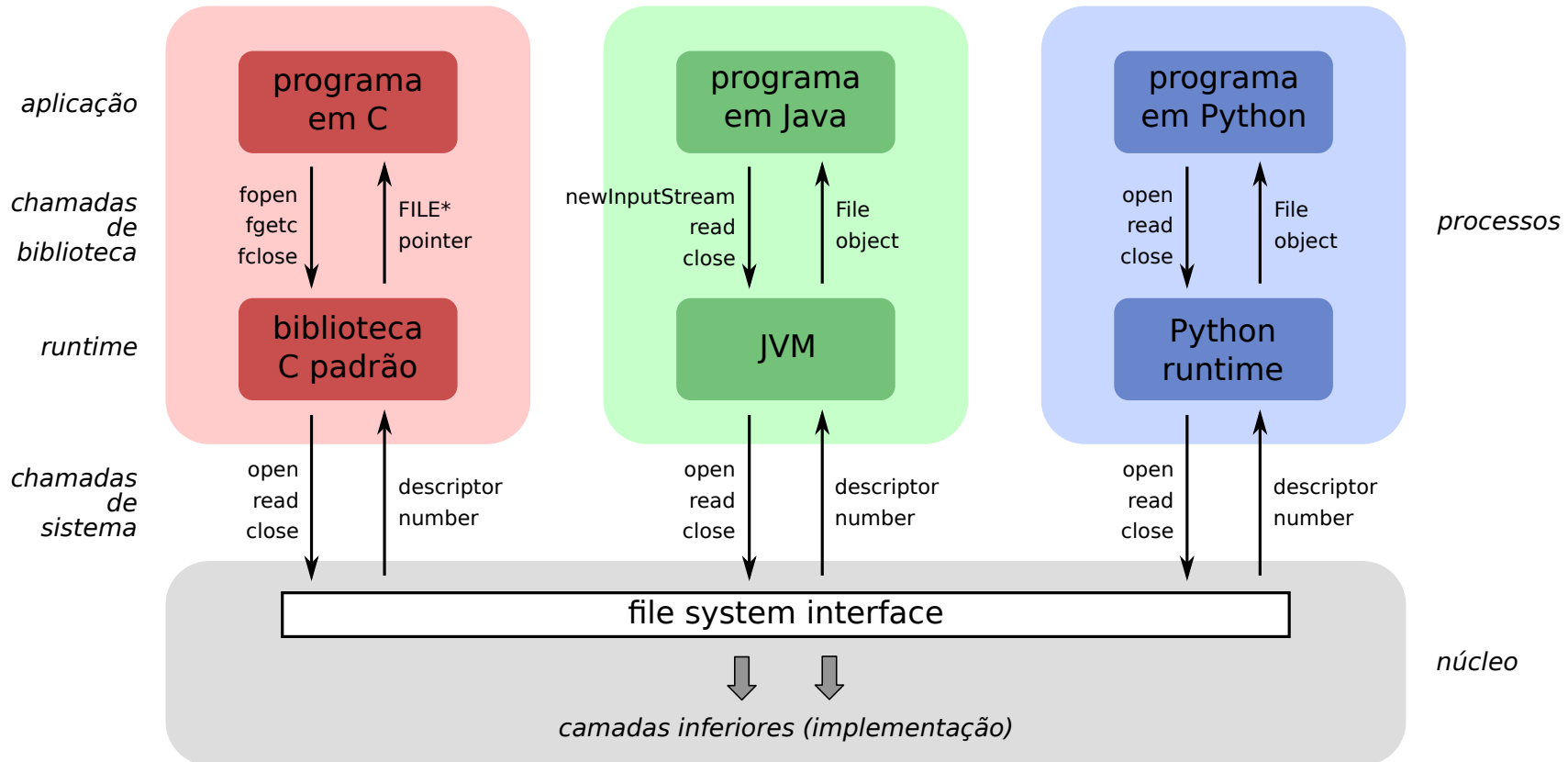
Operações de **alto nível**:

- Chamadas de biblioteca
- Independem do SO, dependem da linguagem

Operações de **baixo nível**:

- Chamadas de sistema
- Dependem do SO, independem da linguagem

Interface de acesso



Chamadas de sistema

Operação	**Linux**	**Windows**
Abrir arquivo	`OPEN`	`NtOpenFile`
Ler dados	`READ`	`NtReadRequestData`
Escrever dados	`WRITE`	`NtWriteRequestData`
Fechar arquivo	`CLOSE`	`NtClose`
Remover arquivo	`UNLINK`	`NtDeleteFile`
Criar diretório	`MKDIR`	`NtCreateDirectoryObject`

Funções de bibliotecas

Operação	**C (padrão C99)**	**Java (classe `File`)**
Abrir arquivo	`fd = fopen(...)`	`obj = File(...)`
Ler dados	`fread(fd, ...)`	`obj.read()`
Escrever dados	`fwrite(fd, ...)`	`obj.write()`
Fechar arquivo	`fclose(fd)`	`obj.close()`
Remover arquivo	`remove(...)`	`obj.delete()`
Criar diretório	`mkdir(...)`	`obj.mkdir()`

Descritor de arquivo

Estrutura que representa um **arquivo aberto** no processo ou no núcleo do sistema

Descritores de **alto nível**, providos pelas funções de bibliotecas:

- C: variável dinâmica do tipo `FILE*`
- Java: objetos da classe `File`
- Python: *file objects*, criados pela chamada **open**

Descritores de **baixo nível**, providos pelas *syscalls*:

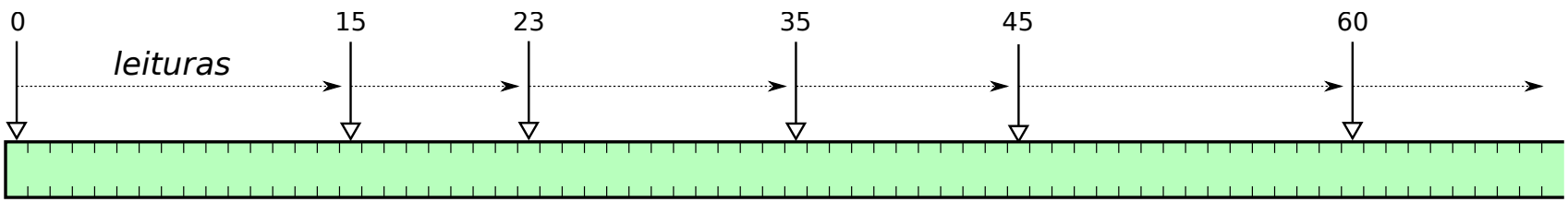
- Windows: *file handle* (*struct* opaco)
- Linux: *file descriptor* (*int*)

A abertura de um arquivo

1. No processo:
    1. a aplicação pede a abertura do arquivo (`fopen()`)
    2. o *runtime* da linguagem invoca uma *syscall*
2. No núcleo:
    1. o núcleo localiza o arquivo no dispositivo físico
    2. verifica as permissões de acesso do arquivo
    3. cria um descritor para o arquivo aberto no núcleo
    4. insere o descritor em uma lista de arquivos abertos
    5. devolve ao processo uma referência ao descritor
3. No processo:
    1. o *runtime* recebe o descritor de baixo nível
    2. o *runtime* cria um descritor de alto nível
    3. a aplicação recebe o descritor de alto nível

Acesso sequencial

- dados acessados em sequência do início ao fim do arquivo
- Para cada arquivo aberto há um *ponteiro de acesso*
- A cada leitura ou escrita, o ponteiro é incrementado
- Um *flag* (*EoF - End-of-File*) indica o fim do arquivo



Acesso direto (ou aleatório)

Acesso direto a posições específicas do arquivo

- Indica-se a posição do arquivo onde fazer o acesso
- `read (file, position, size, buffer)`
- Importante em SGBDs e aplicações similares
- Raramente implementada "pura" nos SOs

Geralmente feita com o reposicionamento do ponteiro:

- chamada de sistema `lseek()`
- chamada de biblioteca C `fseek()`

Mapeamento em memória

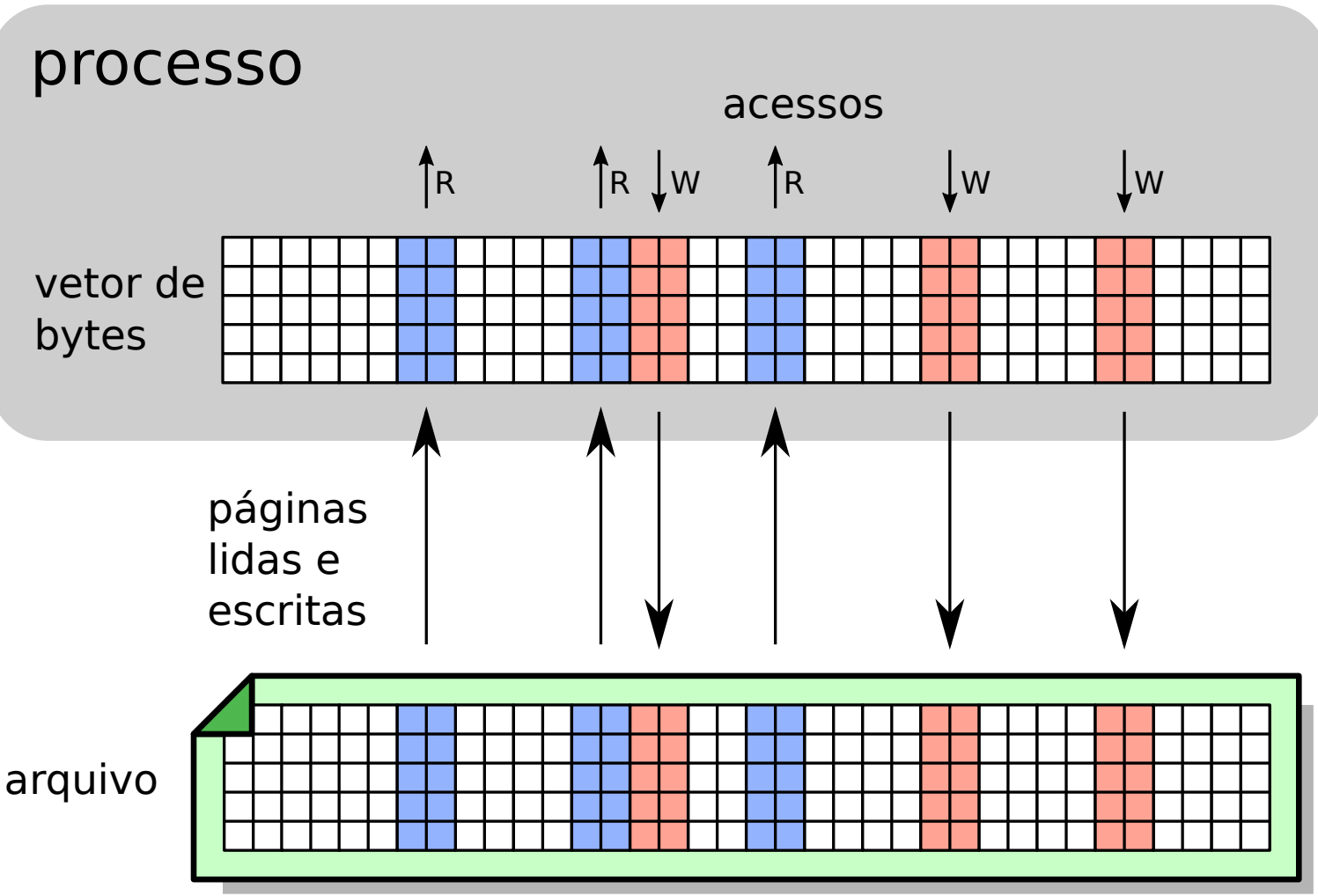
Usa os mecanismos de memória virtual (páginação)

O arquivo é associado a um vetor de bytes em RAM

Quando uma posição do vetor é lida, é gerada uma falta de página e o conteúdo correspondente no arquivo é lido

Podem ser mapeadas apenas partes do arquivo

Mapeamento em memória



Acesso indexado

Implementado em alguns sistemas, como o OpenVMS

Estrutura interna do arquivo é vista com tabela indexada

Operações de acesso usando pares chave/valor:

- `value = read (key)`
- `write (key, value)`

Acesso muito rápido (indexação implementada no núcleo)

Pouco usado nos SOs atuais, mas disponível por bibliotecas

Travas (*locks*)

Controle de concorrência (*mutex*):

- Necessário ao compartilhar arquivos entre processos
- Arquivos ou trechos podem ser travados
- Vários tipos de travas estão disponíveis

Várias travas são oferecidas:

- pelo sistema operacional
- por bibliotecas

Tipos de travas

Trava obrigatória (*mandatory lock*)

- incontornável, imposta pelo núcleo aos processos
- processos são suspensos aguardando liberar a trava
- *default* nos sistemas Windows

Trava recomendada (*advisory lock*)

- gerenciada pelo suporte de execução (biblioteca)
- pode ser ignorada pela aplicação
- útil para gerenciar concorrência dentro de uma aplicação
- o programador deve usar as travas conforme necessário
- *default* nos sistemas UNIX

Tipos de travas

Trava exclusiva (*write lock*)

- garante acesso exclusivo ao arquivo

Trava compartilhada (*read lock*)

- impede travas exclusivas sobre o arquivo
- permite mais travas compartilhadas

Equivalente ao problema dos “leitores/escritores”

Semânticas de compartilhamento

Em acessos concorrentes a um arquivo compartilhado:

- Como cada processo vê os dados do arquivo?
- Como escritas concorrentes se comportam?

Várias possibilidades (“semânticas”):

- Semântica imutável
- Semântica UNIX
- Semântica de sessão
- Semântica de transação

Semânticas de compartilhamento

Semântica imutável:

- Arquivos compartilhados são marcados como **imutáveis**
- Solução simples para garantir a consistência dos dados
- Usada em alguns sistemas de arquivos distribuídos
- Modo default nos sistemas Windows

Semântica UNIX:

- Modificações são visíveis imediatamente a todos
- Usual em sistemas de arquivos locais UNIX

Semânticas de compartilhamento

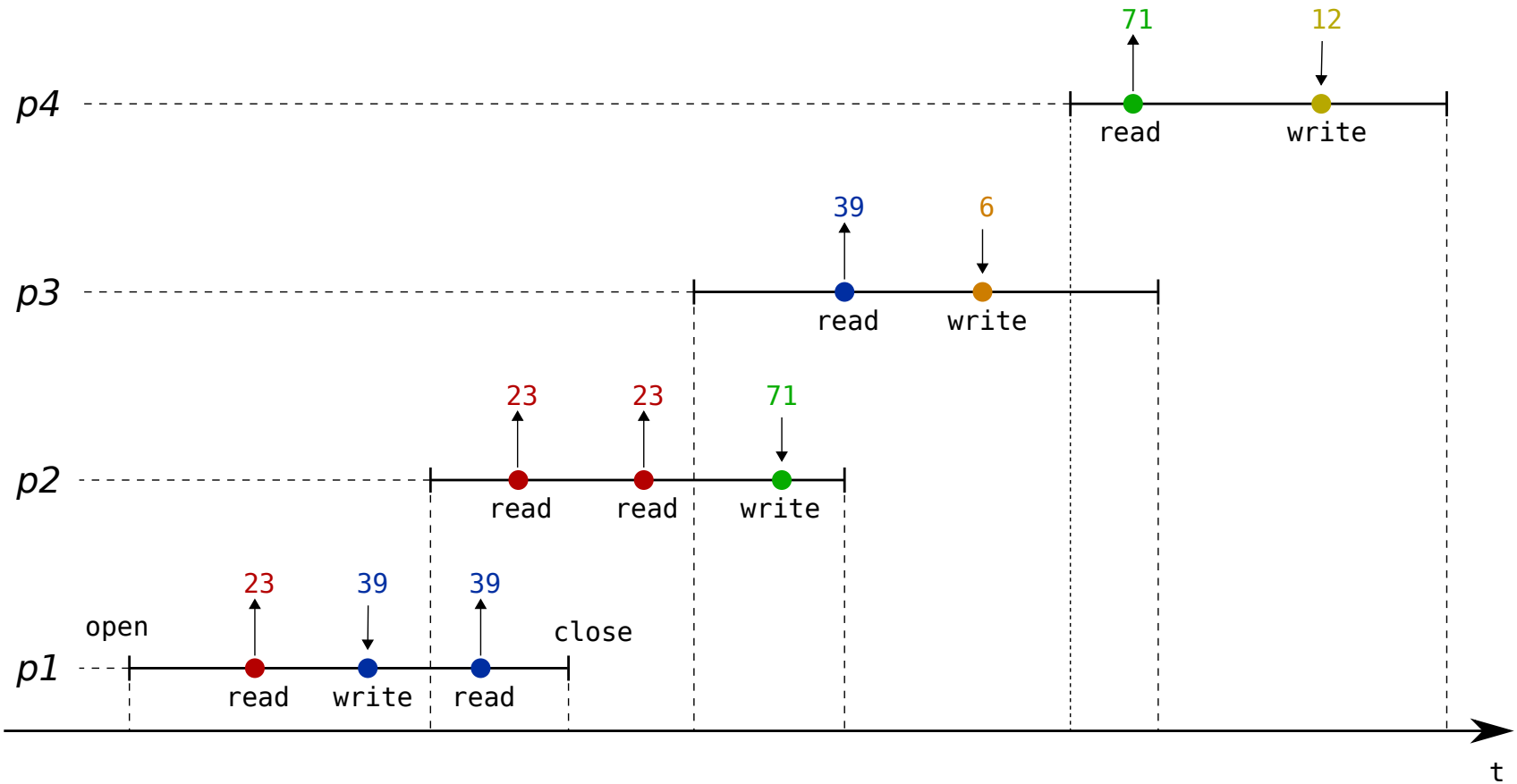
Semântica de sessão:

- cada processo usa um arquivo em uma sessão
- A sessão vai da abertura ao fechamento do arquivo
- Modificações só são visíveis aos demais no fim da sessão
- Sessões concorrentes podem ver conteúdos distintos
- Normalmente usada em sistemas de arquivos de rede

Semântica de transação:

- Similar à seção, mas mais curta (`begin ... end`)
- Usa o conceito de *transação* dos SGBDs

Semântica de sessão



Controle de acesso

Definir e controlar que usuários podem acessar que arquivos

Atributos relevantes:

- **Proprietário**: usuário dono do arquivo
- **Permissões**: operações permitidas aos usuários

Implementação:

- Definição de usuários e grupos
- Listas de controle de acesso (ACL)
- Controle do acesso pelo núcleo do SO
- Modelos mais complexos (SELinux, AppArmor, MIC, etc)

ACL

Access Control List associada a cada recurso

Indica as ações que os usuários podem fazer sobre o recurso

Exemplo:

```
arq1.txt  : (João: ler), (José: ler, escrever), (Maria: ler, remover)
video.avi : (José: ler), (Maria: ler)
musica.mp3: (Daniel: ler, escrever, apagar)
```

Permissões em UNIX

- Usuários:
  - `user` : o proprietário do arquivo
  - `group`: um grupo de usuários associado ao arquivo
  - `other`: os demais usuários
- Permissões: `read`, `write`, `execute`

```
1 $ ls -l
2 drwx------ 2 maziero prof  4096 2008-09-27 08:43 figuras
3 -rwxr-x--- 1 maziero prof  7248 2008-08-23 09:54 hello-unix
4 -rw-r--r-- 1 maziero prof    54 2008-08-23 09:54 hello-unix.c
5 -rw------- 1 maziero prof    59 2008-08-23 09:49 hello-windows.c
6 -rw-r--r-- 1 maziero prof 89580 2008-09-26 22:08 main.pdf
7 -rw------- 1 maziero prof 40494 2008-09-27 08:44 main.tex
```

API de acesso em C

- Abertura e fechamento de arquivos:
  - `fopen (const char *fname, ...)`
  - `fclose (FILE *f)`
- Leitura e escrita de caracteres e *strings*:
  - `fputc (int c, FILE *f)`
  - `fgetc (FILE *f)`
- Uso do ponteiro do arquivo:
  - `ftell (FILE *f)`
  - `fseek (FILE *f, long offset, ...)`
  - `feof (FILE *f)`
- Uso de travas:
  - `flockfile (FILE *f)`
  - `funlockfile (FILE *f)`

API de acesso em C

```c
int main (int argc, char *argv[])
{
   FILE *arq ;                  // descritor de arquivo
   char c ;

   arq = fopen ("infos.dat", "r") ; // abre arquivo para leitura

   if (! arq) {                 // descritor inválido
      perror ("Erro ao abrir arquivo") ;
      exit (1) ;
   }

   c = getc (arq) ;             // le um caractere do arquivo
   while (! feof (arq)) {       // enquanto não chegar ao fim do arquivo
      putchar (c) ;             // imprime o caractere na tela
      c = getc (arq) ;          // le um caractere do arquivo
   }

   fclose (arq) ;               // fecha o arquivo
}
```