

# Sistemas Operacionais

## Conceitos e Chamadas de sistemas

Profª Drª Thaína Aparecida Azevedo Tosta

[tosta.thaina@unifesp.br](mailto:tosta.thaina@unifesp.br)

# Aula passada

- Conceitos iniciais
- O que é um sistema operacional?
- História dos sistemas operacionais
- O zoológico dos sistemas operacionais
- Revisão sobre hardware de computadores

# Sumário

- Processos
- Espaços de endereçamento
- Arquivos
- Entrada/Saída
- Proteção
- O interpretador de comando (shell)
- Chamadas de sistemas
- Chamadas de sistemas para gerenciamento de processos
- Chamadas de sistemas para gerenciamento de arquivos

**Objetivo: conhecer como o SO manipula dados e executa suas tarefas, e questão de fixação.**

# Processos

- Um processo é basicamente um programa em execução associado a:
  - Espaço de endereçamento (programa executável, os dados do programa e sua pilha);
  - Conjunto de recursos (registradores, lista de arquivos abertos, alarmes pendentes, listas de processos relacionados) e todas as demais informações necessárias para executar um programa.
- Exemplo em sistema de multiprogramação: editor de vídeo, navegador e receptor de e-mail (três processos ativos).

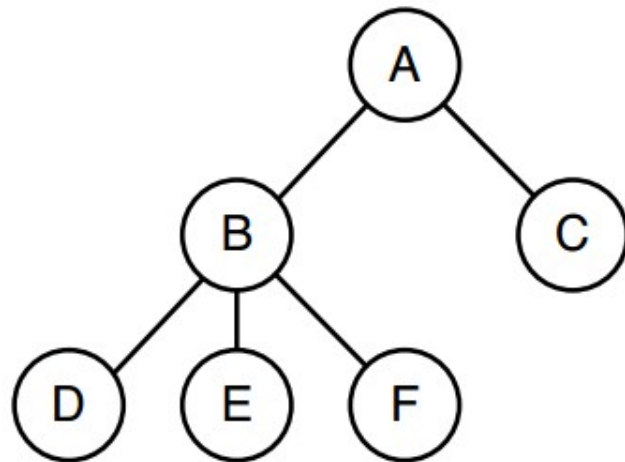
# Processos

- Periodicamente, o sistema operacional decide parar de executar um processo e começa a executar outro;
- Quando isso acontece, todas as informações a respeito do processo precisam ser salvas;
  - **Tabela de processos:** arranjo de estruturas do sistema operacional com uma entrada para cada processo existente.
- Um processo suspenso consiste em seu espaço de endereçamento (imagem do núcleo) e de sua entrada na tabela.

# Processos

## Exemplo:

- Uma pessoa digita no interpretador de comandos (shell) requisitando a compilação de um programa;
- O shell tem de criar agora um novo processo para executar o compilador → **processos filhos**;
- Se os processos cooperam para finalizar uma tarefa, é necessário que eles se comuniquem e sincronizem suas atividades → **comunicação entre processos (sinais, semáforos, pipes).**



# Processos

- Todo processo iniciado tem a UID (*User IDentification*) da pessoa que o iniciou;
- Um processo filho tem a mesma UID que o seu processo pai;
- Usuários podem ser membros de grupos com uma GID (*Group IDentification*);
- A UID de superusuário (em UNIX) ou Administrador (no Windows) sobrepõe-se sobre regras de proteção.

# Espaços de endereçamento

- Todo computador tem alguma memória principal que ele usa para armazenar programas em execução;
- O controle da memória é feito pelo sistema operacional (multiprogramação e proteção);
- O que acontece se um processo tem mais espaço de endereçamento do que o computador tem de memória principal e o processo quer usá-lo inteiramente?
  - Memória virtual (abstração de memória principal + disco criada pelo sistema operacional).



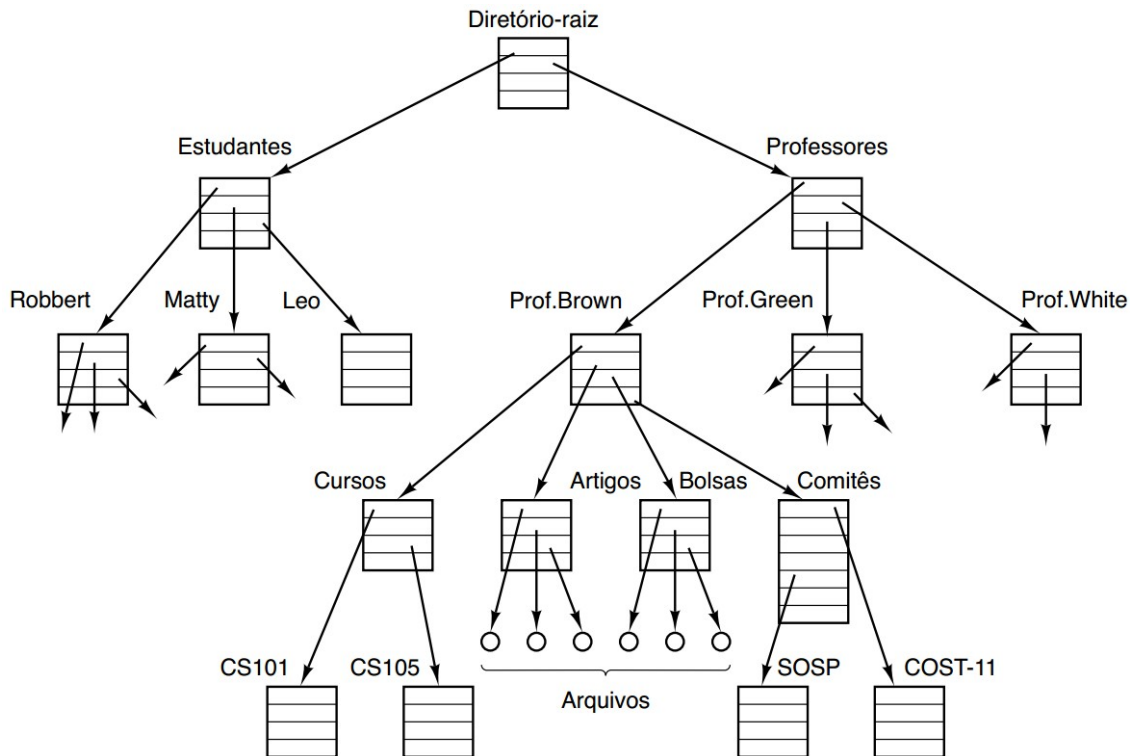
# Arquivos

- O sistema operacional oferece suporte ao sistema de arquivos;
- Para manter e agrupar os arquivos, a maioria dos sistemas operacionais tem o conceito de um diretório;
- Chamadas de sistema são necessárias para (i) criar, remover, ler e escrever arquivos, (ii) criar e remover diretórios, e (iii) mover e remover arquivos de diretórios.

# Arquivos

Entradas de diretório podem ser de arquivos ou de outros diretórios → hierarquia → **sistema de arquivos**.

- Topo: diretório-raiz;
- Nome de caminho absoluto:  
/Professores/Prof.Brown/Cursos/CS101.

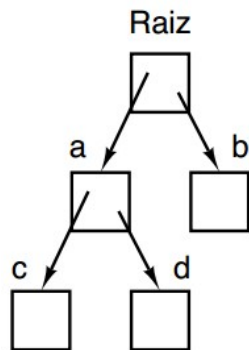


# Arquivos

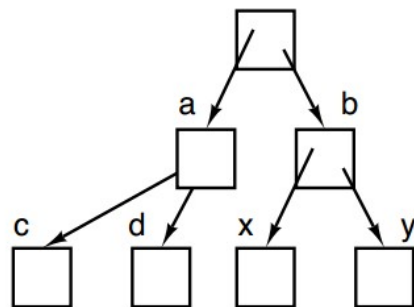
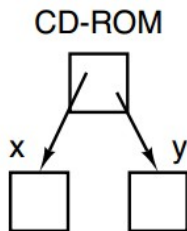
- A todo instante, cada processo tem um diretório de trabalho atual;
- Os processos podem mudar seu diretório de trabalho emitindo uma chamada de sistema;
- Questões de privacidade: descritor de arquivo (pequeno valor inteiro para operações subsequentes).

# Arquivos

- A chamada de sistema mount permite que o sistema de arquivos no CD-ROM seja agregado ao sistema de arquivos-raiz;
- A impossibilidade de acessar arquivos de b não é tão sério porque sistemas de arquivos são quase sempre montados em diretórios vazios.



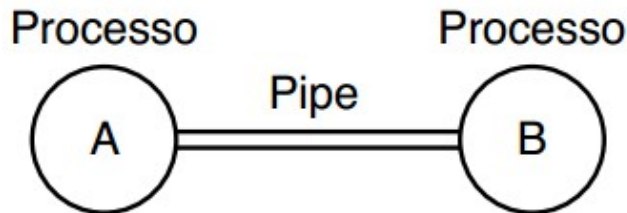
(a)



(b)

# Arquivos

- Relação entre processos e arquivos: **pipes**;
- Um pipe é uma espécie de pseudoarquivo que pode ser usado para conectar dois processos, com implementação parecida com arquivos;
- É necessário configurá-lo antes de usar.



# Entrada/Saída

- Todos os computadores têm dispositivos físicos para obter entradas e produzir saídas;
- Em consequência, todo sistema operacional tem um subsistema de E/S para gerenciar os dispositivos de E/S:
  - Partes independentes dos dispositivos;
  - Partes específicas aos dispositivos (drivers).

# Proteção

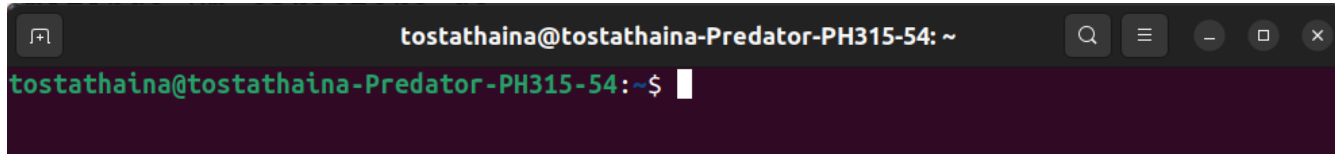
- O sistema operacional também gerencia a segurança do sistema, com autorização de acesso a usuários;
- No UNIX: proteção de arquivos com 9 bits (bits *rwX*):

<i>r</i>	<i>w</i>	<i>x</i>	<i>r</i>	<i>w</i>	<i>x</i>	<i>r</i>	<i>w</i>	<i>x</i>
Proprietário			Membros do grupo do proprietário			Demais usuários		
<i>r</i>	<i>w</i>	<i>x</i>	<i>r</i>	-	<i>x</i>	-	-	<i>x</i>

- Para diretórios, *x* indica busca.

# O interpretador de comando (shell)

- Ainda que o shell não faça parte do sistema operacional, ele faz um uso intenso de muitos aspectos desse sistema e serve como um bom exemplo de como as chamadas de sistema são usadas.
- O shell inicia com um caractere de prompt

A screenshot of a terminal window with a dark background. The title bar at the top shows the user 'tostathaina' and the host 'tostathaina-Predator-PH315-54'. The terminal itself displays the same prompt 'tostathaina@tostathaina-Predator-PH315-54: ~' followed by a green prompt character '\$' and a white cursor. The window has standard Linux window controls (search, menu, zoom, close) on the right side.

```
tostathaina@tostathaina-Predator-PH315-54: ~
```

date (cria processo filho, executa o programa date como um filho e espera sua finalização)

date >file (altera saída padrão)

sort <file1 >file2 (altera entrada e saída padrão)

cat file1 file2 file3 | sort >/dev/lp (pipe)

cat file1 file2 file3 | sort >/dev/lp &



# Chamadas de sistemas

- Qualquer computador de uma única CPU pode executar apenas uma instrução de cada vez;
- Se um processo estiver executando em modo de usuário e precisa de um serviço de sistema, ele tem de executar uma instrução de armadilha (trap) para transferir o controle para o sistema operacional.

`contador = read(fd, buffer, nbytes)`

`fd`: arquivo

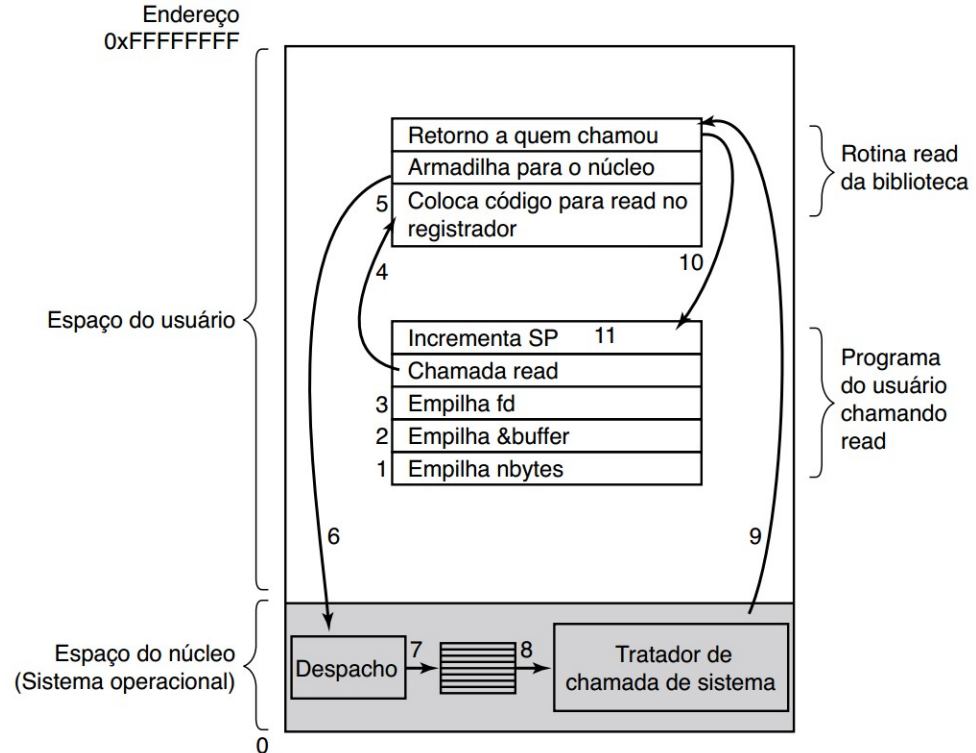
`buffer`: ponteiro para onde os dados devem ser colocados

`nbytes`: qtde de bytes lidos

- Rotina de biblioteca faz a chamada de sistema de mesmo nome.

# Chamadas de sistemas

- 1-3:** Empilha os parâmetros;
- 4:** Chamada para a rotina de biblioteca;
- 5:** Coloca o nº da chamada de sistema onde o sistema operacional o espera;
- 6:** Executa instrução TRAP (execução em endereço físico no núcleo);
- 7:** Despacha para o tratador de chamadas de sistema que geralmente utiliza tabela de ponteiros para as rotinas de tratamento das chamadas de sistema;
- 8:** Execução do tratamento da chamada de sistema;
- 9:** Controle retornado para o espaço de usuário;
- 10:** Retorno para o programa de usuário;
- 11:** Manipulação da pilha.



# Chamadas de sistemas

## Gerenciamento de processos

Chamada	Descrição
<code>pid = fork( )</code>	Cria um processo filho idêntico ao pai
<code>pid = waitpid(pid, &amp;statloc, options)</code>	Espera que um processo filho seja concluído
<code>s = execve(name, argv, environp)</code>	Substitui a imagem do núcleo de um processo
<code>exit(status)</code>	Conclui a execução do processo e devolve status

## Gerenciamento de arquivos

Chamada	Descrição
<code>fd = open(file, how, ...)</code>	Abre um arquivo para leitura, escrita ou ambos
<code>s = close(fd)</code>	Fecha um arquivo aberto
<code>n = read(fd, buffer, nbytes)</code>	Lê dados a partir de um arquivo em um buffer
<code>n = write(fd, buffer, nbytes)</code>	Escreve dados a partir de um buffer em um arquivo
<code>position = lseek(fd, offset, whence)</code>	Move o ponteiro do arquivo
<code>s = stat(name, &amp;buf)</code>	Obtém informações sobre um arquivo

# Chamadas de sistemas

## Gerenciamento do sistema de diretório e arquivo

Chamada	Descrição
s = mkdir(name, mode)	Cria um novo diretório
s = rmdir(name)	Remove um diretório vazio
s = link(name1, name2)	Cria uma nova entrada, name2, apontando para name1
s = unlink(name)	Remove uma entrada de diretório
s = mount(special, name, flag)	Monta um sistema de arquivos
s = umount(special)	Desmonta um sistema de arquivos

## Diversas

Chamada	Descrição
s = chdir(dirname)	Altera o diretório de trabalho
s = chmod(name, mode)	Altera os bits de proteção de um arquivo
s = kill(pid, signal)	Envia um sinal para um processo
seconds = time(&seconds)	Obtém o tempo decorrido desde 1º de janeiro de 1970

# Chamadas de sistemas para gerenciamento de processos

- A chamada **fork** é a única maneira para se criar um processo novo como uma cópia exata do processo original;
- Após a fork, o processo original e a cópia (o processo pai e o processo filho) seguem seus próprios caminhos separados.

`pid = fork()`

Processo filho	Processo pai
0	PID do filho

- Quem é pai e quem é filho? PID!

# Chamadas de sistemas para gerenciamento de processos

- Após uma fork, o processo filho precisará executar um código diferente do processo pai;
  - Exemplo: o shell lê um comando, cria um processo filho, espera que ele execute e então lê o próximo comando quando o processo filho terminar.
- Para esperar que o processo filho termine, o processo pai executa uma chamada de sistema **waitpid**

```
pid = waitpid(pid, &statloc, options)
```

<b>statloc</b>	Com o fim de waitpid, o endereço recebe o estado de saída do filho (término normal ou anormal e valor de saída)
<b>options</b>	Opções diversas, como retornar imediatamente se nenhum processo filho já tiver terminado.

# Chamadas de sistemas para gerenciamento de processos

- Quando um comando de usuário é digitado no shell, ele é executado pela chamada de sistema **execve**;
- Por ela, toda a imagem de núcleo do filho é substituída pelo arquivo nomeado no seu primeiro parâmetro.

`s = execve(name, argv, environp)`

<b>name</b>	Nome do arquivo a ser executado
<b>argv</b>	Ponteiro para arranjo de argumentos
<b>enviropn</b>	Ponteiro para o arranjo de ambiente

# Chamadas de sistemas para gerenciamento de processos

Um interpretador de comandos simplificado:

```
#define TRUE 1

while (TRUE) {
    type_prompt( );
    read_command(command, parameters);

    if (fork( ) != 0) {
        /* Codigo do processo pai. */
        waitpid(-1, &status, 0);
    } else {
        /* Codigo do processo filho. */
        execve(command, parameters, 0);
    }
}
```



# Chamadas de sistemas para gerenciamento de processos

`cp fd1 fd2`: copia `fd1` para `fd2`

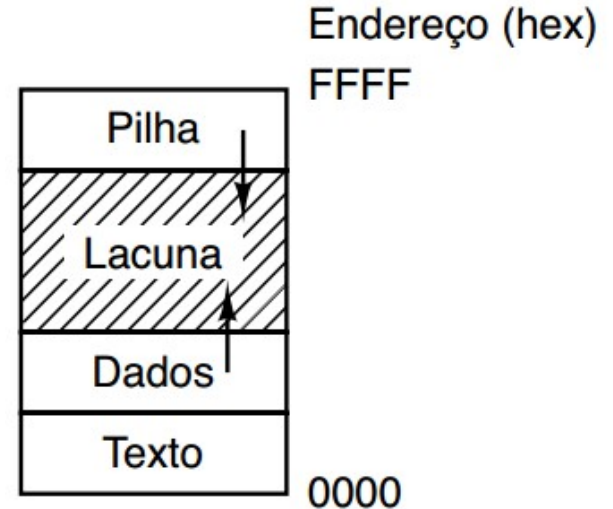
- O shell cria o processo filho;
  - O processo filho localiza e executa o arquivo `cp`, passando os arquivos de origem e de destino.
- O programa principal de `cp`:  
`main(argc,argv,envp)`

<b>argc</b>	contagem de itens na linha de comando (3)
<b>argv</b>	ponteiro para o arranjo na linha de comando ( <code>argv[0]</code> : <code>cp</code> , <code>argv[1]</code> : <code>fd1</code> , <code>argv[2]</code> : <code>fd2</code> )
<b>envp</b>	ponteiro para o ambiente (arranjo de caracteres com informações como o tipo de terminal e o nome do diretório home para programas. Se nenhum ambiente é passado, <code>environp</code> em <code>execve</code> é 0, como no exemplo)

# Chamadas de sistemas para gerenciamento de processos

Processos em UNIX têm sua memória dividida em três segmentos:

- Segmento de texto (código de programa);
- Segmento de dados (as variáveis);
- Segmento de pilha (cresce conforme necessário).



# Chamadas de sistemas para gerenciamento de arquivos

- Muitas chamadas de sistema relacionam-se ao sistema de arquivos;
- Para ler ou escrever em um arquivo, é preciso primeiro abri-lo:

```
fd = open(file, how, ...)
```

<b>file</b>	Nome do arquivo por caminho absoluto ou relativo ao diretório de trabalho
<b>how</b>	O_RDONLY, O_WRONLY, O_RDWR, O_CREAT
<b>...</b>	Definições de permissão
<b>fd</b>	Descritor de arquivos

# Chamadas de sistemas para gerenciamento de arquivos

O arquivo pode ser

- Fechado: `s = close(fd);`
- Lido com armazenamento de dados na memória: `n = read(fd, buffer, nbytes);`
- Escrito pelos dados especificados pelo buffer da aplicação: `n = write(fd, buffer, nbytes);`

# Chamadas de sistemas para gerenciamento de arquivos

- Associado a cada arquivo há um ponteiro que indica a posição atual no arquivo;
- A chamada **lseek** muda o valor do ponteiro de posição, de maneira que chamadas subsequentes para ler ou escrever podem começar em qualquer parte no arquivo

`position = lseek(fd, offset, whence)`

<b>fd</b>	Descritor de arquivo
<b>offset</b>	Posição do arquivo
<b>whence</b>	A posição do arquivo é relativa ao começo, à posição atual ou ao fim do arquivo?
<b>position</b>	Posição absoluta no arquivo (em bytes) após mudar o ponteiro

# Chamadas de sistemas para gerenciamento de arquivos

- Para cada arquivo, UNIX registra o tipo do arquivo (regular, especial, diretório, e assim por diante), tamanho, hora da última modificação e outras informações, acessíveis por

`s = stat(name, &buf)`

<b>name</b>	Arquivo a ser inspecionado
<b>buf</b>	Estrutura na qual a informação deverá ser colocada

- Para um arquivo aberto: `fstat`.

# Chamada de sistema para pipe?

# Objetivo: conhecer como o SO manipula dados e executa suas tarefas e questão de fixação.

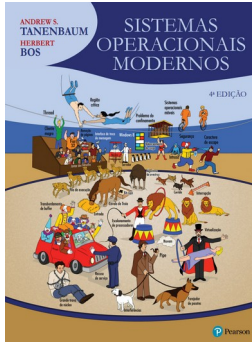
**QUESTÃO 46** – O programa (em linguagem C) abaixo executa em um sistema operacional da família UNIX. Considere que todas as rotinas invocadas no programa executam sem erro. Assinale a alternativa que indica o resultado impresso na tela pelo programa.

```
signed int i;  
int main(void){  
    if ( fork() > 0 )  
        i++;  
    else  
        i++;  
    i++;  
    printf("%d ", i);  
}
```

- A) 1 1
- B) 2 2
- C) 3 3
- D) 4 4
- E) Indeterminado Indeterminado



# Referências



TANENBAUM, Andrew S.; BOS, Herbert. Sistemas operacionais modernos. 4. edição. São Paulo: Pearson, 2016. xviii, 758 p. ISBN 9788543005676.