■ Tema 1: ft_popen (Process I/O Redirection)

O objetivo é criar uma função que inicia um processo e retorna um único descritor de arquivo (file descriptor) para que o processo pai possa ler a saída do filho (tipo 'r') ou escrever na entrada do filho (tipo 'w').

Funções-Chave: `pipe`, `fork`, `dup2`, `execvp`, `close`, `exit`.

"Olá, Filho" (Foco: `fork`)

- **Objetivo:** Entender a duplicação de processos.
- **Explicação:** `fork()` cria um novo processo (filho) que é uma cópia do processo atual (pai). A função retorna `0` para o filho e o PID do filho para o pai.
- **Dica:** Escreva um programa onde o pai imprime "Eu sou o pai" e o filho imprime "Eu sou o filho".
- **Teste:** Verifique se ambas as mensagens aparecem (a ordem pode variar).

"Tornando-se Outro" (Foco: `execvp`)

- **Objetivo:** Entender como um processo se substitui por outro.
- **Explicação:** `execvp(const char *file, char *const argv[])` substitui a imagem do processo atual pelo programa `file`. O `argv` é o array de argumentos (onde `argv[0]` é o nome do programa).
- **Dica:** Crie um programa que chama `execvp("ls", (char *const[]){"ls", "-l", NULL});`.
- **Teste:** Se `printf("Isso não deve ser impresso ");` for colocado após o `execvp`, ele nunca deve ser executado, pois o processo foi substituído.

Executando um Comando Externo (Foco: `fork` + `execvp`)

- **Objetivo:** Executar um comando em um processo filho sem encerrar o processo pai.
- **Dica:** Combine os exercícios 1 e 2. O pai deve usar `fork()`. O filho (`pid == 0`) deve chamar `execvp`.

- **Teste:** Execute seu programa. Ele deve imprimir a saída de `ls -l` e depois terminar.

O Telefone de Copo (Foco: `pipe`)

- **Objetivo:** Criar um canal de comunicação unidirecional.
- **Explicação:** `pipe(int fd[2])` cria um "cano". `fd[0]` é o descritor de arquivo para **ler** do cano. `fd[1]` é o descritor de arquivo para **escrever** no cano.
- **Dica:** Crie um pipe. Escreva a string "oi" em `fd[1]`. Leia de `fd[0]` em um buffer e imprima.
- **Teste:** Verifique se "oi" é impresso.

Comunicação Pai-Filho (Foco: `pipe` + `fork`)

- **Objetivo:** Fazer o pai e o filho conversarem.
- **Dica:** Crie um pipe *antes* de `fork()`. O pai fecha `fd[0]` e escreve em `fd[1]`. O filho fecha `fd[1]` e lê de `fd[0]`.
- **Teste:** O pai envia "ping" e o filho o imprime.

Redirecionamento de Saída (Foco: `dup2`)

- **Objetivo:** Entender como `dup2` remapeia descritores de arquivo.
- **Explicação:** `dup2(int oldfd, int newfd)` faz `newfd` apontar para o mesmo recurso que `oldfd`. Se `newfd` for `STDOUT_FILENO` (que é `1`), qualquer coisa escrita no `printf` ou `write(1, ...)` irá para `oldfd`.
- **Dica:** Abra um arquivo (`open("out.txt", ...)`), e chame `dup2(fd_do_arquivo, STDOUT_FILENO)`. Depois chame `printf("Isso vai para o arquivo")`.
- **Teste:** Verifique o conteúdo de `out.txt`.

Implementando `ft_popen` (tipo 'r', parte 1)

- **Objetivo:** Capturar a saída do filho (`ls`).

Dica: Este é o núcleo do `ft_popen`.

1. `pipe(fd)`
2. `fork()`
3. **Filho:** Fecha a ponta de leitura (`fd[0]`). Redireciona sua saída padrão para a ponta de escrita do pipe (`dup2(fd[1], STDOUT_FILENO)`). Fecha `fd[1]`. Executa o comando (`execvp("ls", ...)`).
4. **Pai:** Fecha a ponta de escrita (`fd[1]`).

- **Teste:** O pai ainda tem `fd[0]`. O que acontece se o pai ler dele?

Implementando `ft_popen` (tipo 'r', parte 2)

- **Objetivo:** O pai deve retornar o FD correto.
- **Dica:** A função do pai no exercício 7 fez tudo certo. Agora, em vez de ler, apenas `return fd[0];`.
- **Teste:** O `main` (que chama `ft_popen`) agora pode ler o `fd` retornado para obter a saída do `ls`.

Implementando `ft_popen` (tipo 'w')

- **Objetivo:** Enviar dados para a entrada do filho (`grep picoshell`).

Dica: A lógica é o inverso do tipo 'r'.

1. `pipe(fd)`
2. `fork()`
3. **Filho:** Fecha `fd[1]`. Redireciona sua *entrada* padrão para a ponta de leitura do pipe (`dup2(fd[0], STDIN_FILENO)`). Fecha `fd[0]`. Executa o comando (`execvp("grep", ...)`).
4. **Pai:** Fecha `fd[0]`. Retorna `fd[1]`.

- **Teste:** O `main` pode escrever "picoshell
" e "outracoisa
" no `fd` retornado. A saída deve ser "picoshell".

Exercício Composto: `ft_popen` Completo

- **Objetivo:** Juntar 'r' e 'w' e gerenciar FDs.

Dica: Sua função `ft_popen` principal deve:

1. Validar os parâmetros (`type` deve ser 'r' ou 'w').
2. Chamar `pipe()`.
3. Chamar `fork()`.
4. **Filho:** Chamar `close()` em *ambos* `fd[0]` e `fd[1]` após `dup2` e antes de `execvp` (o `execvp` falhará se `fd[0]` ou `fd[1]` extras forem deixados abertos em alguns casos).
5. **Pai:** Fechar a ponta *não utilizada* do pipe e retornar a ponta *utilizada*.

- **Teste:** Teste ambos os modos 'r' e 'w' e verifique com `lsof` ou `valgrind` se não há file descriptors vazados.
