■■ Tema 2: picoshell (Process Pipeline)

O objetivo é executar múltiplos comandos (ex: `cmd1 | cmd2 | cmd3`), onde a saída de um se torna a entrada do próximo.

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

Revisão: `ft_popen` (tipo 'r')

- **Objetivo:** Relembrar como capturar a saída de um filho.
- **Dica:** A lógica do `ft_popen` (tipo 'r') é a base para o *primeiro* comando em um pipeline.

Esperando por Crianças (Foco: `wait`)

- **Objetivo:** Aprender a "recolher" processos filhos para evitar zumbis.
- **Explicação:** `wait(int *status)` (ou `waitpid`) pausa o processo pai até que um filho termine.
- **Dica:** Crie um programa que faz `fork()` 3 vezes, e depois o pai chama `wait(NULL)` 3 vezes em um loop.
- **Teste:** O pai só deve terminar *depois* que todos os 3 filhos terminarem.

O Problema de Duas Crianças (`A | B`)

- **Objetivo:** Conceituar o pipeline de dois comandos.

Dica: O que precisa acontecer?

1. `pipe(fd)`
2. `fork()` (para o processo A).
3. Filho A: `dup2(fd[1], STDOUT_FILENO)`, fecha `fd[0]`, `fd[1]`, executa `A`.
4. `fork()` (para o processo B).
5. Filho B: `dup2(fd[0], STDIN_FILENO)`, fecha `fd[0]`, `fd[1]`, executa `B`.
6. Pai: Fecha `fd[0]`, `fd[1]`. `wait()` duas vezes.

- **Teste:** Tente implementar `ls | grep picoshell` usando esta lógica "hardcoded".

O Problema do Loop: A Variável `in_fd`

- **Objetivo:** Generalizar a lógica para N comandos.

- **Dica:** Em vez de `fork`ar todos de uma vez, use um loop. O `picoshell` irá iterar sobre `cmds[]`.

- **Provocação:** Para o `cmd[i]`, a entrada (STDIN) deve vir do `cmd[i-1]`. Precisamos de uma variável para armazenar o "descritor de entrada" da iteração anterior. Vamos chamá-lo de `in_fd`. No início, `in_fd = STDIN_FILENO` (ou `0`).

O Loop do picoshell (Parte 1: O `pipe` e o `fork`)

- **Objetivo:** Esboçar o loop `while (cmds[i])`.

Dica: Dentro do loop:

1. Se *não* for o último comando, crie um novo `pipe(fd)`.
2. Chame `fork()`.

O Loop do picoshell (Parte 2: Lógica do Filho)

- **Objetivo:** O que o processo filho faz.

Dica: Dentro do `if (pid == 0)`:

1. Se `in_fd != 0` (não é o primeiro comando), `dup2(in_fd, STDIN_FILENO)`.
2. Se *não* for o último comando, `dup2(fd[1], STDOUT_FILENO)`.
3. Feche *todos* os FDs que não são mais necessários (o `in_fd` original, `fd[0]`, `fd[1]`).
4. `execvp(cmds[i][0], cmds[i])`.

- **Teste:** Esta lógica é complexa. Teste com `ls | grep picoshell` novamente, mas usando este loop.

O Loop do picoshell (Parte 3: Lógica do Pai)

- **Objetivo:** O que o processo pai faz.

Dica: Dentro do `else // Parent`:

1. Se `in_fd != 0`, feche-o (o filho já o usou).
2. Se *não* for o último comando, feche `fd[1]` (o pai não escreve).
3. **A Mágica:** `in_fd = fd[0]`. A ponta de leitura deste pipe se torna a *entrada* para a *próxima* iteração do loop.
4. Avance `i++`.

Fechando as Pontas Soltas

- **Objetivo:** O último comando.
- **Dica:** O que acontece quando `cmds[i+1]` é `NULL`? O loop não cria um `pipe`. A lógica do filho (Exercício 6) vê que *não* é o último comando é falso, então ele não redireciona o `STDOUT`, que permanece `1` (a saída padrão do terminal). Isso está correto!
- **Teste:** `ls | grep picoshell | wc -l`. Verifique se a saída é `1`.

Esperando Todos (Foco: `wait`)

- **Objetivo:** Recolher todos os filhos após o loop.
- **Dica:** O pai saiu do loop `while (cmds[i])`. Agora ele deve chamar `wait()` em um loop (`while (wait(...) > 0)`) até que todos os filhos tenham terminado.
- **Teste:** Seu `picoshell` deve esperar a pipeline inteira terminar antes de retornar.

Exercício Composto: `picoshell` Completo

- **Objetivo:** Juntar tudo, incluindo o gerenciamento de erros.

Dica: Sua função `picoshell` deve:

1. Inicializar `in_fd = 0` e `i = 0`.
2. Iniciar o loop `while (cmds[i])`.
3. Criar o `pipe` (se não for o último).

4. `fork()`.
5. Lógica do filho: `dup2` de `in_fd` e `fd[1]`, fechar tudo, `execvp`.
6. Lógica do pai: fechar `in_fd` e `fd[1]`, salvar `fd[0]` no *novo* `in_fd`.
7. Após o loop de `fork`, fechar o último `in_fd`.
8. Loop de `wait()`.
9. Retornar `0` em sucesso ou `1` se `pipe` ou `fork` falharem.

- **Teste:** `echo 'squalala' | cat | sed 's/a/b/g'` deve imprimir `squblblb`.
