

Roteiro de Estudo Aprofundado: A Interface de Linha de Comando (Versão Definitiva)

Visão Geral

O objetivo é dominar as ferramentas que criam a experiência de usuário de um shell moderno. Não vamos apenas aprender a ler uma linha, mas a gerenciar um histórico de comandos e a manipular a exibição do prompt de forma dinâmica. Mais importante, vamos resolver o complexo desafio de integrar sinais assíncronos (como `Ctrl+C`) com o loop de leitura síncrono da `readline`, usando as técnicas corretas e seguras.

Módulo 1: O Problema - Por Que a Leitura Simples Não Basta?

(Fundamental e Inalterado)

Exercício 1.1: A Experiência de Usuário Mínima (Reflexão Teórica)

- **Foco:** Análise de Requisitos.
- **Tarefa:** Reflita sobre as limitações de `fgets` para edição de linha e histórico de comandos, solidificando a necessidade da biblioteca `readline`.

Módulo 2: As Ferramentas Essenciais de Interação

(Fundamental e Inalterado)

Exercício 2.1: A Fundação - `readline()`

- **Foco:** Contrato da função `readline()` (alocação de memória, retorno `NULL` em `Ctrl+D`).
- **Tarefa:** Crie `readline_test.c` para ler e liberar a entrada em um loop.

Exercício 2.2: A Memória - `add_history()`

- **Foco:** Função `add_history()`.
- **Tarefa:** Modifique `readline_test.c` para adicionar comandos ao histórico, refinando a lógica para ignorar linhas vazias.

Módulo 3: Gerenciamento do Histórico

(Fundamental e Inalterado)

Exercício 3.1: O Esquecimento - `rl_clear_history()`

- **Foco:** Função `rl_clear_history()`.
- **Tarefa:** Crie `history_test.c` onde um comando específico (`clear_history`) limpa o histórico da `readline`.

Módulo 4: Ferramentas de Manipulação de Interface (UI)

(Fundamental e Inalterado)

Exercício 4.1: O Contexto - `rl_on_new_line()`

- **Foco:** Entender `rl_on_new_line()` como uma função de estado.

Exercício 4.2: A Modificação - `rl_replace_line()`

- **Foco:** Entender que `rl_replace_line()` altera o buffer interno, mas não a tela.

Exercício 4.3: A Renderização - `rl_redisplay()`

- **Foco:** Usar `rl_redisplay()` para forçar a atualização da tela, tornando visíveis as mudanças feitas por `rl_replace_line`.

[MÓDULO ATUALIZADO] Módulo 5: A Integração Avançada - Sinais e o Gancho de Eventos

Objetivo: Entender por que a abordagem simples de verificar uma variável global após `readline` falha, e dominar a técnica correta usando o `rl_event_hook` para criar um shell verdadeiramente responsivo.

Exercício 5.1: O Desafio da Integração e a Falha Esperada

- **Foco:** Compreender o problema do bloqueio.

- **Tarefa (Experimentação):** Tente a abordagem "ingênua". Em um programa de teste, configure um handler de `SIGINT` que define uma flag global `g_signal_status`. No seu `main`, após a chamada a `readline`, verifique `if (g_signal_status == SIGINT)`. Pressione `Ctrl+C`. O que acontece? O `if` é acionado imediatamente ou somente depois que você pressiona Enter?
- **Ponto de Consolidação:** Esta experiência prova que o loop principal fica bloqueado e não pode verificar a flag. `readline` é uma "caixa preta" que não nos devolve o controle. Precisamos de uma maneira de executar nosso código *enquanto* a `readline` está esperando.

Exercício 5.2: A Solução - O Gancho de Eventos (`rl_event_hook`)

- **Foco:** `rl_event_hook` e `rl_done`.
- **Explicação Conceitual:** `rl_event_hook` é um ponteiro para uma função que você fornece. A `readline` se compromete a chamar essa sua função periodicamente enquanto espera pela entrada. Dentro dessa função "gancho", podemos verificar nossa flag de sinal. Se a flag estiver ativa, podemos definir a variável global `rl_done = 1`; da própria `readline`, o que a instrui a parar de esperar e retornar o controle imediatamente para o nosso loop `main`.

Tarefa: Crie um programa `hook_test.c`.

1. Defina a flag global `volatile sig_atomic_t g_signal_status = 0`;
2. Crie um handler de `SIGINT` simples que apenas define `g_signal_status = SIGINT`;
3. Crie a função gancho:

```
c int event_hook(void) { if (g_signal_status != 0) { rl_done = 1; // Força a
readline a retornar } return 0; // 0 para continuar }
```
4. Na `main`, antes do loop, "registre" seu gancho: `rl_event_hook = event_hook`;
5. Execute e pressione `Ctrl+C`. Observe como `readline` agora retorna instantaneamente.

Módulo 6: A Síntese Final - O Shell Responsivo

Objetivo: Combinar o gancho de eventos com as funções de UI para implementar a resposta correta e final ao `Ctrl+C`.

Exercício 6.1 (Composto): A Sequência de `Ctrl+C`

- **Foco:** Combinação de `rl_event_hook`, `g_signal_status`, `write`, `rl_on_new_line`, `rl_replace_line`, `rl_redisplay`.
- **Tarefa (Implementação Final):** Agora que você tem um método 100% funcional para detectar o `SIGINT` imediatamente, implemente a lógica de reação correta dentro do `if (g_signal_status == SIGINT)` no seu loop `main`.
- Descreva e implemente a sequência exata de chamadas necessárias para:
 1. **Resetar a flag:** `g_signal_status = 0;` (Para não entrar neste bloco de novo).
 2. **Mover para a próxima linha:** `write(1, " ", 1);`
 3. **Informar readline da nova posição:** `rl_on_new_line();`
 4. **Limpar o buffer de texto interno:** `rl_replace_line("", 0);`
 5. **Redesenhar o prompt e a linha na tela:** `rl_redisplay();`
- **Ponto de Consolidação:** Ao final deste exercício, seu Minishell terá um tratamento de `Ctrl+C` que é não apenas visualmente idêntico ao `bash`, mas arquiteturalmente robusto e seguro. Você terá dominado a integração entre sinais assíncronos e uma biblioteca de I/O síncrona.