Roteiro de Estudo: Dominando Sinais para o Minishell

Visão Geral

Este roteiro irá guiá-lo desde o conceito mais fundamental de um sinal até a implementação completa e segura no seu projeto. Cada exercício é um bloco de construção. Ao final, você não terá apenas o código, mas o domínio completo do "porquê" por trás de cada linha.

Módulo 1: Fundamentos - O Que é um Sinal?

Objetivo: Observar e entender o comportamento padrão de um processo ao receber sinais, sem qualquer tipo de tratamento.

Exercício 1.1: A Vítima

- **Foco:** Observação.
- **Tarefa:** Crie `victim.c` com um loop infinito e `sleep(1)`. Envie `SIGINT` (`Ctrl+C`) e `SIGQUIT` (`Ctrl+\`). Anote a ação padrão para cada um. Este comportamento é aceitável para o Minishell?

Módulo 2: O Básico da Captura - A Função `signal()`

Objetivo: Aprender a forma mais simples de registrar um handler para entender o conceito de "captura".

Exercício 2.1: Captura com `signal()`

- **Foco:** Função `signal()`.
- **Tarefa:** Crie `signal_test.c`. Escreva um handler que usa `write`. Na `main`, use `signal(SIGINT, handler);`. Mantenha o programa rodando. Pressione `Ctrl+C`. O que mudou?

Módulo 3: O Padrão Moderno - A Família `sigaction`

Objetivo: Dominar a API POSIX moderna, dissecando cada função e sua responsabilidade.

Exercício 3.1: Captura com `sigaction()`

- **Foco:** Função `sigaction()`.
- **Tarefa:** Crie `sigaction_test.c` para replicar o exercício 2.1 usando `struct sigaction` e `sigaction()`.

Exercício 3.2: Inicializando a Máscara com `sigemptyset()`

- **Foco:** Função `sigemptyset()`.
- **Tarefa:** Modifique `sigaction_test.c`, adicionando `sigemptyset(&sa.sa_mask);` antes de configurar o handler. Este é um passo de "higiene" de código.

Exercício 3.3: Bloqueio Específico com `sigaddset()`

- **Foco:** Função `sigaddset()`.
- **Tarefa:** Crie `mask_test.c`. Crie um handler para `SIGINT` que dorme por 3 segundos. Na sua configuração, use `sigaddset(&sa_int.sa_mask, SIGQUIT);`. Enquanto o handler de `SIGINT` estiver executando, pressione `Ctrl+\`. Observe quando o sinal `SIGQUIT` é de fato tratado.

Exercício 3.4 (Composto): Configurando Múltiplos Sinais

- **Foco:** Combinação das ferramentas `sigaction`.
- **Tarefa:** Crie `multi_handler.c`. Configure seu programa para: 1. Ignorar `SIGQUIT` (`SIG_IGN`). 2. Capturar `SIGINT` com um handler customizado. 3. Deixar `SIGTSTP` (`Ctrl+Z`) com seu comportamento padrão. Teste cada um.

[NOVO] Módulo 4: O Handler Informativo e a Comunicação Inter-Processos

Objetivo: Aprender a usar o modo avançado da `sigaction` para que o handler receba informações detalhadas sobre o sinal, incluindo o PID do processo remetente. Isso é fundamental para a comunicação entre o shell e os processos que ele cria.

Exercício 4.1: Ativando o Modo Informativo

- **Foco:** A flag `SA_SIGINFO` e a nova assinatura do handler.

Tarefa: Crie um programa `info_test.c`.

1. Escreva uma função handler com a nova assinatura: `void info_handler(int signum, siginfo_t *info, void *ucontext)`. Por enquanto, faça-a apenas imprimir uma mensagem genérica com `write`.

2. Na `main`, ao configurar sua `struct sigaction`, adicione a flag: `sa.sa_flags = SA_SIGINFO;`.

3. Em vez de usar `sa.sa_handler`, você **deve** usar o membro `sa.sigaction`: `sa.sa_sigaction = info_handler;`.

4. Registre este handler para `SIGUSR1`. Mantenha o programa rodando e envie um sinal `SIGUSR1` de outro terminal usando `kill -SIGUSR1 <PID>`. Verifique se seu novo handler é chamado.

Exercício 4.2: Identificando o Remetente

- **Foco:** O campo `si_pid` da estrutura `siginfo_t`.

Tarefa: Modifique `info_test.c`.

1. Dentro do seu `info_handler`, acesse o PID do remetente através de `info->si_pid`.

2. Use `write` para imprimir na tela uma mensagem como: "Sinal recebido do processo com PID: [número do pid]". (Você precisará de uma função auxiliar como `ft_putnbr_fd` para converter o PID para uma string imprimível).

3. Para testar, você precisará de dois terminais:

- **Terminal 1:** Execute `./info_test` e anote seu PID.

- **Terminal 2:** Anote o PID do próprio terminal (`echo $$`). Agora, envie o sinal do Terminal 2 para o processo no Terminal 1: `kill -SIGUSR1 <PID_do_info_test>`.

4. Observe a saída no Terminal 1. O PID impresso deve corresponder ao PID do seu shell no Terminal 2.

Módulo 5: O Padrão de Comunicação Segura (Para `Ctrl+C`)

Objetivo: Implementar o padrão `handler -> global variable -> main loop`, que é a arquitetura correta para lidar com sinais como `SIGINT` no modo interativo.

Exercício 5.1: A Ponte Atômica

- **Foco:** `volatile sig_atomic_t`.
- **Tarefa:** Crie `safe_comm.c`. Declare a variável global, crie um handler que *apenas* modifica essa variável, e um loop `main` que *apenas* lê a variável e age de acordo, resetando-a em seguida.

Módulo 6: Integração Final com o Projeto Minishell

Objetivo: Aplicar todo o conhecimento adquirido no ambiente real do seu projeto.

Exercício 6.1: Preparando o Terreno em `signals.c`

- **Foco:** Organização do código do projeto.
- **Tarefa:** Crie `src/signals.c` com a variável global, o handler simples (para `SIGINT`), e a função `setup_signals(void)` que configura `SIGINT` e ignora `SIGQUIT`.

Exercício 6.2: O Tratamento da Interrupção em `main.c`

- **Foco:** Lidar com o retorno de `readline` após um sinal.
- **Tarefa:** Modifique `src/main.c`. Chame `setup_signals()`. Dentro do loop, após `readline`, verifique a variável global e, se necessário, resete-a.

Exercício 6.3 (Final): Restaurando o Prompt

- **Foco:** Funções da biblioteca `readline` para UI.
- **Tarefa:** Melhore o bloco condicional do exercício anterior. Use a sequência correta de `write(" ", 1)`, `rl_on_new_line()`, `rl_replace_line("", 0)`, e `rl_redisplay()` para imitar o comportamento do `bash` ao receber `Ctrl+C`.

Se você seguir este roteiro de **10 exercícios em 6 módulos**, terá construído seu conhecimento do básico ao avançado, incluindo comunicação inter-processos, culminando na solução robusta e correta exigida pelo seu projeto.