Tarefa de Codificação: API de Processos

June 10, 2025

Introdução

Esta tarefa de codificação visa proporcionar familiaridade com as APIs de gerenciamento de processos em sistemas UNIX. Você escreverá programas em C para explorar o comportamento de chamadas de sistema como `fork()`, `exec()`, `wait()`, e outras. Certifique-se de compilar e executar os programas em um ambiente UNIX/Linux para testar os resultados.

Questões

1. Escreva um programa que chama `fork()`. Antes de chamar `fork()`, faça o processo principal acessar uma variável (por exemplo, `x`) e definir seu valor (por exemplo, 100). Qual é o valor da variável no processo filho? O que acontece com a variável quando tanto o filho quanto o pai alteram o valor de `x`?

2. Escreva um programa que abre um arquivo (com a chamada de sistema `open()`) e depois chama `fork()` para criar um novo processo. Tanto o filho quanto o pai podem acessar o descritor de arquivo retornado por `open()`? O que acontece quando eles escrevem no arquivo concorrentemente, ou seja, ao mesmo tempo?

3. Escreva outro programa usando `fork()`. O processo filho deve imprimir `"hello"`; o processo pai deve imprimir `"goodbye"`. Você deve tentar garantir que o processo filho sempre imprima primeiro; você consegue fazer isso sem chamar `wait()` no pai?

4. Escreva um programa que chama `fork()` e depois chama alguma forma de `exec()` para executar o programa `/bin/ls`. Veja se consegue experimentar todas as variantes de `exec()`, incluindo (no Linux) `execl()`, `execle()`, `execlp()`, `execv()`, `execvp()`, e `execvpe()`. Por que você acha que existem tantas variantes da mesma chamada básica?

5. Agora escreva um programa que usa `wait()` para esperar que o processo filho termine no pai. O que `wait()` retorna? O que acontece se você usar `wait()` no filho?

6. Escreva uma leve modificação do programa anterior, desta vez usando `waitpid()` em vez de `wait()`. Quando `waitpid()` seria útil?