Insper

16 - chamadas de sistema para gerenciar processos

Igor Montagner

Parte 1 - fork

A chamada `fork` cria um clone do processo atual e retorna duas vezes: uma vez no processo original (pai) e uma vez no processo novo (filho). Cada processo segue executando o programa linha a linha, porém cada um possui áreas de memória separadas. Ou seja, mudar uma variável no processo pai não muda seu valor no filho (e vice-versa). Todo processo é identificado por um número chamado de `pid`.

Exercício: O programa abaixo termina? Explique sua resposta.

```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main() {
    int rodando = 1;
    pid_t filho;

    filho = fork();

    if (filho == 0) {
        printf("Acabei filho\n");
        rodando = 0;
    } else {
        while (rodando) {
            printf("Esperando o exec acabar!\n");
            sleep(1);
        }
    }
    return 0;
}
```

Leia o código abaixo e responda.

```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main() {
    pid_t pai, filho;
    int variavel = 5;

    filho = fork();
    if (filho == 0) {
        // processo filho aqui
        pai = getppid();
        filho = getpid();
        variavel *= 2;
        printf("eu sou o processo filho %d, meu pai é %d\nvariavel %d\n",
            filho, pai, variavel);
    } else {
        // processo pai aqui!
        pai = getpid();
        printf("eu sou o processo pai %d, meu filho é %d\nvariavel %d\n",
            pai, filho, variavel);
    }
    return 0;
}
```

Exercício:

1. ao rodar o programa, qual seria o valor de `variavel` no print do pai? e do filho?
2. esse valor muda conforme o pai (ou o filho) executam primeiro?

Rode o programa (arquivo *exemplo1-fork.c*) e veja se sua saída corresponde as suas respostas acima.

Exercício: faça um programa que cria 8 processos filhos (numerados de 1 a 8) e faz cada um imprimir na tela seu seu identificador. O processo pai deve imprimir 0, enquanto o primeiro filho imprime 1, o segundo 2 e assim em diante. A saída de seu programa deverá seguir o modelo abaixo:

```
Eu sou o processo pai, pid=%d, meu id do programa é %d\n
Eu sou um processo filho, pid=%d, ppied=%d, meu id do programa é %d\n
```

A primeira linha só deve ser mostrada uma vez pelo processo pai. Para verificar que seu programa funciona corretamente não se esqueça de contar quantos `printf` foram feitos. Se houver mais que 9 houve algum problema na sua solução.

Exercícios complementares

Nestes próximos exercícios vamos juntar processos e arquivos.

Exercício: pesquise como usar a chamada `nanosleep` para suspender a execução de um processo por um certo número de tempo. Escreva abaixo os cabeçalhos a serem importados e como chamar a função para dormir por dois segundos.

Exercício: modifique seu `exemplo_io2.c` da última aula para dar `nanosleep` antes de cada `write`. Faça uma versão que dorme por 500ms e uma que dorme por 300ms. Abra dois terminais e rode ambos os programas. Escreva abaixo o que aconteceu.

Este é um exemplo de **concorrência por recursos**. O mesmo recurso é usado simultaneamente e os processos acabam interagindo de maneira descontrolada e estragando o arquivo. Uma solução comum adotada por muitos programas é criar um arquivo `nome.lock` quando iniciam o trabalho com um arquivo e deletá-lo após finalizar

os acessos. Assim, se outras instâncias do mesmo programa tentam acessar o arquivo elas podem detectar a existência do arquivo `.lock` e mostrar uma mensagem de erro.

Exercício: modifique seu `copy_file` para que ele tenha este comportamento. Ou seja, antes de abrir `arquivo` para escrita ele checa se `arquivo.lock` existe e, caso isso seja verdade, mostre uma mensagem de erro. Quais flags devem ser usadas na abertura de `arquivo.lock` para que não existam problemas de concorrência?

Parte 2 - wait, waitpid

Um processo pode esperar seus filhos acabarem usando uma das chamadas `wait` ou `waitpid`. Esta chamada retorna um código numérico que representa a saída do programa (o que foi retornado pelo `main`) ou um conjunto de flags que indica se houve término anormal.

O código **errado** do último exercício tentava simular estas chamadas usando uma variável `rodando` e checando seu valor. A maneira correta de esperar um processo filho terminar é usando `wait` ou `waitpid`.

Exercício: pesquise como usar `wait` no manual. Escreva abaixo a assinatura da função. Qual é o valor retornado? O que é retornado na varíavel passada como ponteiro?

Exercício: Modifique o programa `exemplo2-errado.c` para usar `wait` para esperar o processo filho terminar. Após o filho terminar o pai deve mostrar uma mensagem na tela indicando este fato. Salve este arquivo como `exemplo2-certo.c`

Exercício: É possível obter o valor retornado pelo `main` de um processo usando `wait`. Modifique o `exemplo2-certo.c` para que o filho retorne `2` e modifique o pai para que ele obtenha esta informação a partir dos valores retornados pelo `wait`. Você precisará ler o manual de `wait` para fazer este exercício.