Sistemas Operacionais - Compilando código C no terminal do Linux

Aluno: *Danillo Francisco Leite* - **Matrícula:** *1821145*

Engenharia de Software – Centro Universitário de ANÁPOLIS – UniEvangélica.
Anápolis – Go – Brasil
danillofranleite@gmail.com

1. Introdução

Fork é uma função que é uma chamada de sistema. Ou seja, ela invoca o sistema operacional para fazer alguma tarefa que o usuário não pode. No caso, o fork é usado para criar um novo processo em sistemas do tipo Unix, e isso só pode ser feito via fork.
Quando criamos um processo por meio do fork, dizemos que esse novo processo é o filho, e processo pai é aquele que usou o fork.
Por exemplo, suponha que você programou um software em C, e nele usou a chamada **fork()**. Esse programa em C, executando, é o processo pai.
Quando usamos o fork, será criado o processo filho, que será idêntico ao pai, inclusive tendo as mesmas variáveis, registros, descritores de arquivos etc. Ou seja, o processo filho é uma cópia do pai, exatamente igual.
Porém, é uma cópia, e como tal, depois de criado o processo filho, ele vai ser executado e o que acontece em um processo não ocorre no outro, são processos distintos agora, cada um seguindo seu rumo, onde é possível mudar o valor de uma variável em um e isso não irá alterar o valor desta variável no outro processo.

2. Como usar a função fork()

Além das bibliotecas que normalmente usamos para programar em C, necessitaremos de duas **sys/types.h** e **unistd.h** do Unix, para podermos trabalhar com a fork.

Bibliotecas

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

Para a execução desse código, utilizamos o laboratório de informática e o sistema operacional Linux na máquina virtual Oracle Virtual Box. Após a criação do arquivo com a extensão **.c**, abrimos o terminal do sistema e para compilarmos o código, é necessário digitar alguns comandos para a execução do mesmo.

Comandos

- ***dc caminho do arquivo:*** Esse comando acessa a pasta/diretório onde o arquivo está salvo.
- ***ls:*** Comando para listar todos os conteúdos do diretório.
- ***gcc arquivo.c -o arquivo:*** O gcc é o executável que será criado, o arquivo.c é o código (código fonte do seu programa), e por final, o -o arquivo é seu arquivo de saída.
- ***./arquivo:*** Executa o programa criado. (Obs.: Para a execução do mesmo, é necessário colocar o mesmo nome inserido no comando **gcc... -o arquivo**. Caso contrário, o código não será compilado e também exibirá uma mensagem de erro.).

3. Resultados

No código abaixo, foi utilizado a função **fork()**, na qual é usada para realizar uma tarefa que o usuário não pode. Logo abaixo, será exemplificado um código C em que é utilizada essa função.

Código 1

```
int main(){
    fork();
        printf("\n\n\tPai\n\n");
}
```

Ao visualizar esse código utilizando a função **fork()** em sua execução, será criado um processo filho idêntico ao processo pai, inclusive tendo as mesmas variáveis e etc. Realizando todos os comandos para o compilação do arquivo no terminal Linux, é perceptível em sua prática que, a função **fork()** cria processos exponencialmente. Ou seja, se o usuário declarar essa função duas vezes antes do **printf** (comando utilizado para mostrar valor de qualque tipo), ele terá em sua execução quatro processos filhos criado. Visto que, os processos são iguais ao processo pai.
Para mudarmos a prática dessa compilação modificamos o código C, no qual o mesmo passa ter uma condição e também um laço de repetição para ser analisado sua compilação e execução. Veja abaixo o código:

Código 2

```
for (i=0; i<=4; i++){
    if(fork() ==0)
        printf("\n\n\t\tFilho %i\n", i);
    else
        printf("\n\n\t\tPai %i\n", i);
}
```

No laço de repetição, foi dado um início e fim para a repetição do bloco, onde o mesmo será repetido de 0 até em algum número menor ou igual a 10. Partindo agora para a condição, ao compilar o código no terminal do Linux, temos como resultado uma sequência de processos criados fora de ordem. Isso ocorre porque ao compilar o código com essa código e a utilização da função **fork()**, o resultado tende a mostrar aqueles processos em sua execução devido ao escalonador e a prioridade do processo.

4. Referências

Link do Repositório: https://github.com/Danillofran/Tannus
Nome dos Arquivos: fork_1.c, fork_2.c