UNIVERSIDADE FEDERAL DE OURO PRETO
CIÊNCIA DA COMPUTAÇÃO

ROBSON NOVATO LOBÃO - 20.1.4018

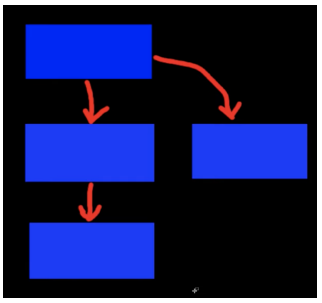
TP2, SISTEMAS OPERACIONAIS - THREADS E PROCESSOS

Ouro Preto
2022

O intuito desse trabalho é entender o funcionamento das threads e processos e como elas conversam entre si para melhor funcionamento do código.
A proposta é que, vamos ter um processamento de saldo que vai ocorrer "ao mesmo tempo".

Na questão 1, a motivação é fazer de tal forma que vamos ter 3 processos rodando concomitantemente, um que vai realizar o print do saldo do usuário e outros dois filhos (repartições do processo original) que vão lidar com o saldo em tempo real, seja somando 100 ou subtraindo 100. Para bom funcionamento do código precisamos ligar esses processo e a melhor forma de fazer isso é através de uma pipe, que em C, é unidirecional e é formado por um vetor de duas posições em que a posição 0 é responsável por ler e a 1, por escrever na pipe, essa informação é transmitida para outro processo e a pipe se esvazia, funcionando como um FIFO (primeiro a entrar é o primeiro a sair), podendo também ocorrer um entupimento da pipe, quando as informações fornecidas não são lidas por ninguém.

Tendo isso em vista, o processo da questão 1 se inicia inicializando as pipes e adicionando à pipe que vai operar o saldo o valor de 0, além de dar dois forks(), ou seja repartimos o projeto inicial em mais 1, e depois os 2 processo em mais 2 da seguinte forma:



A linha logo abaixo da principal (a mais a cima) são as de operação e a mais a direita é a de printar.

Consultando em qual fork, ou processo estamos, é possível realizar a operação desejada pelo usuário, se e somente se o comando informado e pego com o scanf seja correspondente a o esperado, caso não a pipe da operação joga para outro processo já que não era aquele que ele pertencia, então em pseudocódigo seria algo assim:

Estamos no comando pai?
Se sim, leia o que o usuário digitar:
O usuário digitou para printar o saldo?
Se sim
Leia a pipe do saldo
printe o saldo
Se não
procure em outro processo
e transmita a operação desejada via pipe
Estamos no comando filho de soma?
Se sim, leia a operação do pipe
A operação do pipe corresponde à soma?
Se sim,
Leia o saldo através da pipe do saldo
Some o valor ao saldo
Escreva o valor ao saldo da pipe
Encerre e leia outra operação

E basicamente assim para a subtração também. Em todas essas operações é printado a sua pid, ou seja o identificador do seu processo, para ter certeza que não estão coincidindo e conflitando. Se o comando for o correspondente para sair, damos Kill() nos processos filhos e finalizamos a execução.

Com relação à questão 2 o feito é mais fácil, apenas lemos o comando do usuário e dependendo do que ele almeja criar uma thread para somar, subtrair ou printar o saldo e logo após o join() para poder esperar a execução, e quando quiser finalizar, damos exit em todas as threads criadas. Lembrando que também é utilizado o recurso de pegar o tid da operação para informar a thread que estamos trabalhando.

Com relação a diferenciação das funções pthread_join, pthread_kill e pthread_exit temos:

pthread_join: Simplesmente espera a thread acabar seu processo.
pthread_kill: Envia um sinal a thread podendo retornar 0 ou -1 em caso de falha, tem a funcionalidade de informar algum fato à thread.
pthread_exit: Finaliza a thread que estava sendo realizado o processo.

E respondendo a primeira pergunta, quais diferenças encontradas quando fazemos o fork(), ou seja quando usamos processos diferentes e quando usamos threads diferentes. Primeiramente se usarmos um getpid() nas threads criadas, elas vão ser as mesmas e isso acontece pois, os processos podem usar várias threads mas uma thread só roda um processo. Mas a mais gritante é, quando usamos o fork() uma nova área intocada é criada para trabalho, o que significa que, se alterarmos uma variável lá dentro, sem uso de pipes, ela vai se manter alterada apenas lá, pois é um

outro escopo, enquanto que variáveis alteradas em threads diferentes refletem apenas no processo principal, ou seja, threads diferentes compartilham a mesma memória em todos os aspectos.

Link do vídeo com a explicação:
https://youtu.be/ELy35NyiYAk