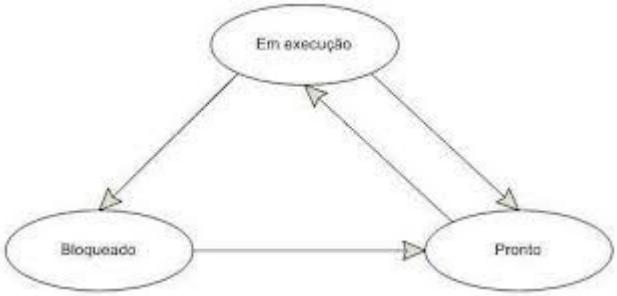
UFOP-DECOM-BCC264 Nº 02/2023-2

2º TP 2023-2

Falamos sobre processos e threads na nossa aula. Processos são "às vezes" apelidados como processos pesados pois tem um espaço de memória próprio, tem um PCB (*Process Control Block*) próprio e um PID (Process Identifier). Em Linux, use o ps-aux e veja os processos rodando. No Windows, o gerenciador de tarefas te dará a relação de quais processos estão sendo executados. Todos os processos listados estão abertos e são executados "um pedaço" no tempo.



O diagrama acima é muito parecido com o diagrama que falamos na última aula. Veja estes slides aqui https://slideplayer.com.br/slide/367578/ .

Para criar um processo, o Unix definiu a primitiva fork "cria" um processo novo. Na verdade, a função fork duplica o processo atual dentro do sistema operacional. Veja este texto: http://www.br-c.org/doku.php?id=fork

Como **processos** não compartilham memória, para comunicarem entre sí, se faz necessário algum mecanismo de IPC (*inter process Communication*) como troca de mensagens (via sockets) ou estabelecer um pipeline.

Threads é uma tarefa (uma linha de execução) que um determinado programa realiza. O processador vai lendo as instruções da memória principal e vai executando as instruções lidas sequencialmente, uma por uma. Lembre-se da arquitetura *von Neumann*? O Contador de Programa (PC) define uma linha de execução onde a instrução é lida na posição de memória apontada pelo registrador PC, decodificada, executada (a instrução), o valor do PC é incrementado e começa tudo de novo. É assim que os programas são executados! Para ter vários "threads" (linhas de execução, isto é, vários programas em execução simultaneamente) rodando em um único computador com 1 processador, basicamente, deve-se "simular" o funcionamento de vários "PCs", onde se tem, na realidade, apenas um. Na prática, a ideia é compartilhar, não só o PC

mas todos os registradores internos, no tempo (cada um executa um pouco, alternativamente). Isto é, um programa executa durante um tempo, e quando o programa é interrompido temporariamente para outro executar, o estado do programa que está executando é salvo (armazenado), carrega-se o estado de um outro processo que foi salvo anteriormente e troca-se o "espaço de memória", para que seja exatamente os mesmos endereços e mesmo estado quando o processo foi salvo anteriormente e... pronto!!! Desta forma, um único processador pode rodar vários programas "simultaneamente", mas veja que, na verdade, não é "simultaneamente" mas parece que é e, na prática, é como se fosse! Por isto, muitas vezes, considera-se uma aplicação "multithread" como uma aplicação paralela pois, na prática, se é um verdadeiro paralelismo (vários processadores /hardware) ou um falso paralelismo ("pseudo paralelismo" usando o mecanismo de executa e para.., como exemplificado acima), pouco importa.

Assim, cada processo tem uma thread (uma linha de execução) por *default*. Porém, muitas vezes se faz necessário que um programa tenha mais de uma linha de execução. Por exemplo, um programa procedimental (feito na linguagem C, por exemplo) pode ter vários procedimentos. Por que não executá-los concorrentemente, compartilhando os recursos do processo "pesado" (espaço de memória, PCB e o todos os outros recursos do processo "pesado")?

E quais as vantagens? Várias, mas cito apenas uma: cada procedimento transformado em um thread será executado *assincronamente*. Lembre-se dos fundamentos da programação de computação? Cada instrução é executada sequencialmente onde uma instrução só é executada após a anterior ter finalizada (executada). Isto é, os programas executavam instruções síncronas!!!

Um processo "pesado" (aquele que é gerado por um "fork", veja abaixo) tem apenas uma thread, por *default*. Mas, pode ter mais e para isto foi idealizado a biblioteca pthread (a biblioteca "raiz em C"). Então, se o processo "pesado" tiver só uma thread.. você não precisa se preocupara com nada.. é só programar, compilar e colocar para executar. Mas, se você quiser (e necessitar) que seu programa tenha procedimentos que sejam executados concorrente ("simultaneamente" e assincronamente), aí você verá como threads é útil. Toda linguagem de programação (que executa em SOs modernos) implementa ou tem uma biblioteca "padrão" que implementa algo similar o que a biblioteca pthreads do "C" faz (provavelmente com outro nome).

Veja http://www.br-c.org/doku.php?id=threads_posix

Da mesma forma que os processos sofrem escalonamento, os threads também têm a mesma necessidade. Quando vários processos são executados em uma CPU, eles dão a impressão que estão sendo executados simultaneamente. Com as threads ocorre o mesmo, elas esperam até serem executadas. Como esta alternância é muito rápida, há impressão de que todas as threads são executadas paralelamente.

Observe que não há nenhum comando que detalha cada tid (*thread identifier*) como o ps-aux (do Linux/unix).

1.Seu TP2

Imagine que você tem uma variável chamado SALDO, inicializada com zero. Toda vez que você apertar a tecla mais, que iremos representar como "+", irá incrementar 500 Unidades de Dinheiro (chamaremos UD, certo?) e cada vez que vc apertar "-" irá decrementar 500 UDs.

Você deverá apresentar o saldo e as instruções para incrementar e decrementar o saldo. Se o usuário apertar "+" o saldo incrementará 500UD e o "-"decrementará (subtrairá) 500 UD.

Porém, você implementará das seguintes formas:

Usando processos "pesados":

1) Você usará o fork e gerará, no minimo, 3 processos "pesados": 1 que imprimirá o valor do SALDO outro processo incrementará e outro decrementará o mesmo valor. Se quiser, pode "ler o teclado" em outro processo, não importa.. mas os processos SALDO, incrementar e decrementar são obrigatórios. Coloque a opção de "exit" que, quando acionada, destruirá todos os processos anteriormente criados.
   - Imprima ("printe") o número do processo (pid)
   - Use pipe para o IPC (veja http://www.inf.ufes.br/~rgomes/so_fichiers/aula14.pdf)

Usando threads:

2) A mesma coisa acima só que em vez de 3 processos, você vai criar 3 threads. Isto, usará a pthreads_create (em C) e gerará 3 threads: 1 que imprimirá o valor do SALDO outro thread incrementará e outro decrementará. Se quiser, pode "ler o teclado" em outro thread, não importa.. mas os threads SALDO, incrementar e decrementar são obrigatórios. Imprima ("printe") o número do thread (tid) Coloque a opção de "exit" que, quando acionada, destruirá todos os threads anteriormente criados. -> ***Usando pthreds, mostre a diferença entre pthread_join e pthread_kill e pthread_exit*****.**

Gere (Built) uma imagem no Docker Hub de tal forma que eu possa baixar a imagem (veja o TP1)

O que deve entregue:

Entregáveis:

a. Um vídeo mostrando o funcionamento no máximo 5 minutos. Mostre que você que fez, como é solução, que voce entendeu e dominou o assunto e respondeu as questões abaixo,
b. Defina, no video, o que é RACE CONDITION e faça que esta condição aconteça, na prática;
c. Mostre a diferença entre o FORK e o Threads
d. coloque o INDEX nos comentários ou em um arquivo PDF com o tempo onde eu encontro as respostas para as questões acima
e. O "endereço" da imagem, no Docker Hub, do seu trabalho (são dois programas, mas só uma imagem), Favor não baixar a imagem e colocar no Moodle que eu não considero, ok?

POR FAVOR, NÃO ZIPE

1) Preferencialmente poste os links do vídeo e, alternativamente, os vídeos. Tanto faz.
2) A data e hora do deadline do trabalho está no Moodle Então, se adiante para não ter problemas com o Moodle, Google meet,etc..

FINALMENTE, ENTENDA O QUE VOCÊ ESTÁ FAZENDO, pois isto tudo é material de prova.