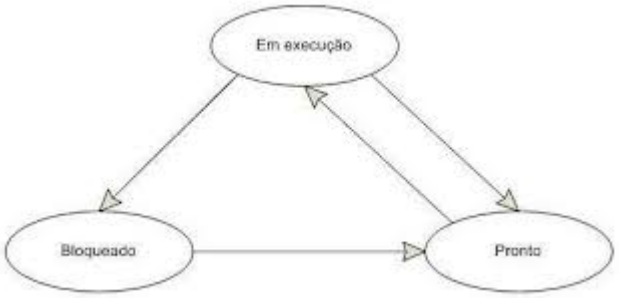
UFOP-DECOM-BCC264 Nº 02/2020-2

2º TP 2020-2
para ~~08-06-2021~~ estendido para o dia 10(quinta)-13h20

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.

Assim, cada processo tem uma thread (uma linha de execução). 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”) ?

Assim, um processo “pesado” tem uma thread, por *default*. Mas, pode ter mais e para isto entra 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 para execução.. Mas, se você quiser (e necessitar) que seu programa tenha procedimentos que sejam executados concorrente, aí você verá como threads é útil. Toda linguagem de programação (que executa em SO modernos) implementa ou tem uma biblioteca “padrão” que implementa threads (pode ter até 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 100 Unidades de Dinheiro (chamaremos UD, certo?) e cada vez que vc apertar “-“ irá decrementar 50 UDs

Você deverá apresentar o saldo e as instruções para incrementar e decrementar o saldo. Se o usuário apertar “+” o saldo incrementará 100UD e o “-“decrementará (subtrairá) 50DUD.

Porém, você implementará das seguintes formas:

Usando processos “pesados”:

1) Você usará o fork e gerará 3 processos “pesados”: 1 que imprimirá o valor do SALDO outro processo incrementará e outro decrementará. 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.**

Faça seu programa de tal forma que eu possa “colar” e “copiar” no https://www.onlinegdb.com/online_c_compiler . Assim, TESTE antes para ver se está tudo certo.

O que deve entregue:

Entregáveis:

a. 1 texto explicando - as diferenças que você encontrou entre threads e “fork” – considerações e – como rodar no https://www.onlinegdb.com/online_c_compiler
b. O vídeo mostrando o funcionamento no máximo 2 minutos. Mostre que entendeu e dominou o assunto e respondeu o texto acima;
c. arquivo em formato PDF de texto explicando o que foi feito;

POR FAVOR, NÃO ZIPE

1) Preferencialmente poste os links do vídeo e, alternativamente, os vídeos. Tanto faz.
2) O trabalho é para o dia 08, terça, mas **foi estendido** para dia 10, quinta, às 13h20. Mas, não será estendido 1 minuto. O sistema fechará e NÂO será aceito nenhuma postagem posterior ou via e-mail Então, se adiante para não ter problemas com o Moodle, Google meet,etc..

© 2021, Prof. Dr. Carlos Frederico M.C. Cavalcanti
DECOM/ICEB/UFOP