Trabalho Prático - Implementação com Pthreads

Grupo: Ana Amélia Traversi, Gustavo da Silva Machado, Gabriel Almeida Gomes, Lucas Zanusso Morais

Biblioteca:

A biblioteca simone.h foi designada para programação multithreading em C e C++. Ela oferece quatro primitivas básicas: start(), spawn(), sync() e finish().

int start(int m);

O start() receberá
Ela recebe como parâmetro um valor inteiro que indica o número de threads a serem inicializados.
Criar o thread, inicializa as variáveis globais e chama o loop que itera.
Ela retornará 1, no caso de sucesso da inicialização, e 0, no caso de falha.
Intrínsico á execução do start(), possuímos a função loop_que_itera();

void* loop_que_itera(void*p);
Essa função usa o do-while para executar pelo menos uma tarefa.
Ele pega as tarefas da lista de acordo com o escalonamento de fila, first-in first-out, que seleciona a tarefa mais antiga. Depois, ele adquire o mutex para a lista de Tarefas Prontas e verifica se ela está vazia. Caso estiver, ele entrará num estado de espera e liberará o mutex para que outro thread possa adquirí-lo.
Como o thread vai acordar de novo? Com um sinal, um signal, do spawn. E ai ele continua a execução, pegando a próxima tarefa, a tarefa mais antiga e retorna o mutex.

Ele pega tarefas da lista.
Política de escalonamento é uma fila, first-in first-out, que pega a tarefa mais antiga da lista.
Ele adquire o mutex para a lista de tarefas prontas e verifica se a lista de tarefas está vazia.
Se estiver, ele entra em estado de espera e libera o mutex para que outro thread possa adquirir o mutex.
Se o thread está trancado e acorda, mas a execução já terminou, ele libera o mutex e retorna zero. Ai que está o paralelismo, porque as tarefas podem ser feitas ao mesmo tempo.
O loop então coloca a tarefa na lista de tarefas terminadas.

int spawn(Attrib *attr = NULL, void* (*func) (void*) = NULL, void* dta = NULL);
O spawn() irá adicionar novas tarefas na fila.
Ela receberá um ponteiro para a classe atributo, um ponteiro para uma função e um ponteiro para os parâmetros necessários.
No início, possuímos um trycatch para verificar se foi enviada alguma função. Não enviar uma função não é um problema em si, mas não será possível fazer um sync desta maneira - a função entrará num loop infinito.
O spawn() dará um lock na lista de Tarefas Prontas e criará um ID, que será retornado. Este ID é um inteiro utilizado pelo sync().

O spawn() gerará um sinal para acordar uma única thread. Este thread será utilizado no loop_que_itera para retirar uma tarefa da lista, executá-la e inserí-la na lista de Tarefas Terminadas.

int sync(int idTarefa, void retornoTarefa);**
O sync() receberá um ID para a tarefa a ser sincronizada e a a posição de memória que está apontando para o retorno da tarefa.
No sync(), consideram-se três casos: a tarefa adquirida está “terminada”, ela está “pronta” ou está em “execução”.
No primeiro caso, ele irá percorrer a lista de Tarefas Terminadas em busca do ID. Se o sync() achá-lo, ele retirará o ID da lista e dar o retorno.
Caso o sync() não encontrá-lo lá, ele irá procurar na lista de Tarefas Prontas, executará a tarefa e não será preciso colocar o ID na lista de Tarefas Terminadas.
Caso a tarefa esteja em execução, o sync() executará outra tarefa e depois voltará para verificar a anterior - até encontrar o ID na lista de Tarefas Terminadas e executar o primeiro caso.

void finish();
O finish() irá anunciar que terminou a execução, liberando todos os threads da lista Tarefas Prontas até a lista esvaziar. Ele também irá resetar as variáveis globais e destruir os mutex e condições do processo.

Execução:

Em nosso trabalho, as primitivas destacam-se por:
- A possibilidade de dar start e finish quantas vezes quisermos.
- Nossa abordagem é como a falta de parâmetros e retornos.
- O fato é que não desperdiçamos CPU e tampouco memória, pois apenas 1 thread é acordado de cada vez.
- O uso de variáveis de condição.

As limitações do código incluem:
- A impossibilidade de iniciar o sync() em uma tarefa que não existe. Só é possível chamá-lo por meio de um ID retornado pela primitiva spawn.
- Não é possível dar dois start() em sequência.

O comportamento do código é imprevisível quando:
- É dado um finish() sem um start()
- O finish() é utilizado em sequência um do outro.
-

Extras:

Não implementamos nenhum outro algoritmo ou função de escalonamento.