Computação Concorrente

Segundo Trabalho

Rafael Balzana e Lucas Rufino

1) Ideia Geral da Solução

Foi utilizada uma fila como estrutura de dado principal para impedir a inanição das threads, garantindo que serão executadas na ordem de chegada.

Todas as threads, em cada iteração, serão primeiramente inseridas na fila, e em seguida irão checar se estão no topo da fila(condição 1). Se não estiverem em primeiro lugar, serão bloqueadas até que outra thread(a que está em primeiro lugar) saia da fila, e execute um broadcast para a condição de topo de fila.

Esse processo será repetido até que a thread esteja no primeiro lugar, e então, no caso de ser uma thread leitora, irá checar se há alguma thread escritora escrevendo, e no caso de ser uma thread escritora, se há qualquer thread escrevendo ou lendo(condição 2).

Caso não haja, ela sairá da fila, incrementando o número de threads lendo ou escrevendo(dependendo de seu tipo) e executando um broadcast para a condição de topo de fila.

Caso haja, ela será bloqueada pela variável da condição 2 até que o número de threads lendo ou escrevendo torne-se zero, recebendo um signal quando a última terminar de executar, e então ocorrerá o processo descrito no caso anterior.

Finalmente ela lerá ou escreverá, dependendo de seu tipo, e então decrementará o número de threads lendo ou escrevendo, e caso esse número torne-se zero, executará um signal da segunda condição mencionada.

2) Documentação

Primeiramente, a seguir está a maneira de passar os argumentos para a execução do arquivo “trabalho.c”

./trabalho.out <numero_de_thread_leitoras> <numero_de_threads_escritoras> <numero_de_leituras_por_thread> <numero_de_escritas_por_thread> <nome_do_arquivo.c>

Estarei agora descrevendo a documentação das duas bibliotecas criadas para a implementação do trabalho. São elas “fila_thread.h” e “funcoes_log.h”.

Documentação da “fila_thread.h”:

Esta biblioteca cuida das funções necessárias para a implementação da fila na solução. Temos as seguintes funções, com suas respectivas documentações:

Tipos:

NodeFila. É a estrutura criada para os nodes que entram na fila de threads. Possuem três campos, sendo eles um ponteiro “ant” para o NodeFila anterior na fila, um int “tid” armazenando o id da thread e um char “tipo” armazenando o tipo da thread (‘L’ para leitor e ‘E’ para escritor).

ThreadFila. É a estrutura criada para a fila de threads. Possui dois campos, sendo eles um ponteiro “primeiro_da_fila” para o primeiro node da fila e um ponteiro “ultimo_da_fila” para o último.

Funções:

ThreadFila *CreateThreadFila();. É o método de criação da fila. Ele retorna um ponteiro para uma fila vazia, ou seja, com ambos os campos nulos.

int *QueueThreadFila(ThreadFila *tf, int id, char tipo);. É o método de inserção da fila. Ele recebe como entrada um ponteiro "tf" para a fila onde se deseja inserir um novo termo. Também recebe um inteiro "id" e um char "tipo" correspondendo aos dois campos que serão inseridos no novo node da fila.

A função então tenta criar um novo node com um malloc. Se o malloc der errado o programa é encerrado com uma mensagem de erro. Se o método der certo a função retorna zero.

int *DequeueThreadFila(ThreadFila *tf);. É a função de remoção da fila. Ela recebe um ponteiro "tf" para a fila onde se deseja remover o termo do topo. O método retorna -1 se a fila estiver vazia. Senão, o termo do topo é removido e é retornado zero.

Int *DestroyThreadFila(ThreadFila *tf);. É o "free" da fila. A função recebe um ponteiro "tf" para a fila onde se deseja desalocar a memória. O método então percorre todos os nodes da fila e desaloca suas respectivas memórias, para então desalocar a memória do próprio ponteiro da fila. É retornado zero se a operação foi concluída com sucesso.

int GetIdTopoDaFila(ThreadFila *tf);. A função recebe um ponteiro "tf" de fila como entrada e retorna o campo "id" do termo do topo da fila. Se a fila estiver vazia é retornado -1.

char GetTipoTopoDaFila(ThreadFila *tf);. A função recebe um ponteiro "tf" de fila como entrada e retorna o campo "tipo" do termo do topo da fila. Se a fila estiver vazia é retornado '\0'.

Documentação da "funcoes_log.h":

Esta biblioteca cuida das funções necessárias para a implementação do arquivo de log do problema. Possui funções implementadas para a validação das restrições propostas no trabalho. As funções são chamadas pelo próprio arquivo de log, visto que ele é um ".c".

Int EntrouNaFila(char tipo, int id, ThreadFila *tf, int *i);. Essa função controla a entrada sequencial das "threads" na fila. Ela incrementa o contador de linhas "i" e insere na fila "tf" o termo caracterizado pelo tipo e id fornecidos na entrada.

Int SaiuDaFila(char tipo, int id, ThreadFila *tf, int *i, int *threads_leitoras_executando, int *thread_escritora_executando);. Primeiro, o contador de linhas "i" é incrementado. Essa função controla a saída sequencial das "threads" na fila. Note que uma "thread" só pode sair da fila se ela for a primeira da fila. Por isso o algoritmo checa se a thread caracterizada pelo id e tipo fornecidos na entrada corresponde à thread do todo da fila "tf". Se não for, o programa encerra com uma mensagem de erro.

Também é checado se a thread está saindo da fila desrespeitando as regras de leitor/escritor. Ou seja, se um escritor está saindo da fila enquanto há um número positivo de leitores/escritores executando, ou se um leitor sai da fila quando há um número positivo de escritores executando. Ambos os parâmetros de leitores e escritores executando são representados respectivamente pelos parâmetros "threads_leitoras_executando" e "thread_escritora_executando" fornecidos na entrada. Se alguma dessas regras for descumprida o método encerra o programa com uma mensagem de erro.

Em seguida, se todas as regras foram respeitadas, o programa incrementa o número de threads executando correspondente ao tipo da thread que acabou de ser removida da fila. E é retornado zero.

Int Executou(char tipo, int id, ThreadFila *tf, int *i, int *threads_leitoras_executando, int *thread_escritora_executando);. Primeiro, o contador de linhas “i” é incrementado. Essa função controla o término de execução sequencial das “threads”. Por isso, dependendo do tipo da thread fornecido no parametro “tipo”, é decrementada a variável “threads_leitoras_executando” ou “thread_escritora_executando”. E então é retornado zero.

3) Casos de teste

OBS:Interpretamos que o número de leituras/escritas passados na entrada se referem ao número executado por cada thread, ou seja, se for passado 5 como número de leituras, cada thread leitora lerá 5 vezes, garantindo balanceamento da tarefa entre elas.

a) 4 threads leitoras, 4 threads escritoras, 20 leituras, 20 escritas

b) 2 threads leitoras, 4 threads escritoras, 25 leituras, 50 escritas

c) 4 threads leitoras, 2 threads escritoras, 200 leituras, 100 escritas

(os arquivos correspondentes a cada caso estão no repositório)