Transações Bancárias com Fila de Requisições com Threads

Arthur F. H. Schuelter[1], Marcelo N. Paolillo[1]

[1]Departamento de Ciência da Computação – Universidade do Estado de Santa Catarina (UDESC)

arthur.schuelter@gmail.com, marcelonunespaolillo@gmail.com

1. Introdução

Este relatório revela como foi implementado o programa, que simula transações bancárias utilizando um número genérico de threads. Além disso, o programa foi desenvolvido na linguagem C e foi projetado para ser executado em computadores com sistema operacional Linux, mais especificamente, Ubuntu.

Para realizar o processamento paralelo de transações bancárias, foi utilizada a API Pthreads. Através dessa, utilizam-se pthreads para retirar elementos da fila de requisições e para debitar e creditar o montante para as contas envolvidas. Para isso, utilizam-se mutexes para limitar o acesso das threads na fila e para impedir a mudança simultânea do saldo das contas envolvidas. Além disso, utiliza-se a biblioteca *semaphore.h* para ordenar o acesso das threads na função de retirada de elementos da fila.

Também foi utilizada a biblioteca fornecida pelo professor, *utils.h*, no qual foi utilizada a função *rand_sleep()* para o processo principal e para as threads.

2. Estruturas de Dados

Para armazenar as contas bancárias, criou-se um vetor em que cada elemento do vetor é uma *struct* do tipo *Conta*. Cada elemento do vetor possui uma variável inteira com número da conta (*unsigned int* numConta), uma variável de precisão dupla para o saldo da conta (*double* saldo) e um mutex (*pthread_mutex_t* lock). Para acessar esse vetor, criou-se um descritor - um ponteiro do tipo DescritorVetor, chamado de dVetor - que posseui o tamanho do vetor e um ponteiro do tipo *Conta*.

Para criar esse vetor de Contas, primeiramente, o programa lê o arquivo de texto com as informações das contas e armazena o número de contas no descritor do vetor. Quando chega no fim do arquivo, realiza a alocação de memória utilizando o tamanho armazenado no descritor, utiliza a função *rewind()* e lê novamente arquivo, colocando as informações das contas no vetor de Contas.

Para criar a fila de requisições, foi implementada uma fila dinâmica duplamente encadeada que possui elementos do tipo *Transferencia*. Nessa *struct*, armazena-se o número de conta do remetente (*unsigned int* numOrigem), o número de conta do destinatário (*unsigned int* numDestino), o montante da transação (*double* quantidade) e dois ponteiros do tipo transação que apontam para as posições anterior (*struct transferencia** anterior) e posterior (*struct transferencia** proximo;) da fila.

O primeiro elemento da fila de requisições pode ser acessado pelo descritor, chamado de dFila, um ponteiro do tipo *DescritorFila*, que possui o tamanho da fila (*int* tamFila) e um ponteiro do tipo Transferência que aponta para o primeiro elemento da fila (*Transferencia** inicio).

3. Fluxo do Programa

Primeiramente, inicializam-se os dois semáforos globais, *pthread_mutex_t* mutex e *sem_t* semaphore. Então, utilizamos a função *void* criaContas(*char** arqcontas) para realizar a leitura do arquivo, fazer a locação de memória do vetor de Contas e popular esse vetor com as informações das contas bancárias.

Antes de a fila de requisições ser instanciada, criamos as threads que aguardam em no semáforo semaphore, contido na função *void* realizaTransferencias(*void** arg), até que existam elementos na fila para serem processados. Para criar a fila de requisições, utiliza-se dois ponteiros auxiliares do tipo *Transferencia*, o aux e o builder. Enquanto o builder serve para criar elementos do tipo *Transferencia*, o aux aponta em qual posição da fila esse elemento criado deve ser inserido. Quando o builder termina de criar um elemento da fila, ele trava o mutex global, insere o elemento aonde o aux aponta e, em seguida, destrava o mutex.

Um elemento só recebe as informações da transação se a conta do remetente e a conta de destinatário existirem. Para isso, criou-se a função *unsigned int* findConta(*unsigned int* conta) que recebe o número da conta e retorna a posição dessa no vetor de contas. Se a conta não existir, o valor de retorno é -1.

Quando um elemento é inserido com êxito, antes de desbloquear o mutex, aumenta o valor de semaphore e o valor de transfeitas, o contador do total de transações. Quando o primeiro elemento é inserido com sucesso, aumenta-se o valor de semaphore, para que as threads possam acessar a fila de requisições.

Quando o programa termina de ler o arquivo das transferências, a variável global flagFim recebe 1, e isso sinaliza para todas as threads que, se a fila estiver vazia, as threads devem sair da função. A verificação de flagFim e do início da fila é feito após a thread passar do semáforo semaphore e após travar o mutex global, se as condições de fim de programa forem satisfeitas, destrave-se o mutex, aumenta o semáforo, a thread sai da função realizaTransferencias e é aguardada com o pthread_join, na main.

Enquanto as condições de fim não forem satisfeitas, as threads acessam a fila de requisições depois de travar o mutex global, apontam o ponteiro aux - do tipo *Transferencia* – para a primeira posição da fila, remove esse elemento da fila de requisições e destrava o mutex global. Assim, essa thread pode processar a transferência, enquanto outra thread acessa a fila de requisições ou o processo principal adiciona elementos na fila.

4. Controle de Concorrência

Foram observados quatro pontos críticos: a inserção de elementos na fila através do processo principal, a remoção de elementos da fila pelas threads, o acesso das threads na fila e a atualização dos saldos das contas após a realização da transação.

Para resolver o problema da inserção e remoção de elementos da fila, foi utilizado uma variável global do tipo pthread_*mutex_t*, chamada de mutex. Assim, sempre que o processo principal ou qualquer uma das threads tentasse acessar a fila, antes é necessário travar o mutex, inserir/remover o elemento da fila, e assim que o processo ou a threads parar de mexer na fila, desbloqueia-se o mutex para que os outros possam acessá-las.

Para resolver o problema de acesso das threads na fila, utiliza-se um semáforo contador, o *sem_t* semaphore, antes que a thread possa travar o mutex e acessar a fila de requisições. Sempre que um novo elemento é inserido na fila, aumenta o semáforo, com a função sem_post(&semaphore), e sempre que uma thread passa pelo semáforo e se aproxima de travar o mutex para poder acessar a fila, abaixa-se o semáforo, com a função sem_wait(&semaphore).

Para que as threads possam atualizar os saldos das contas sem interferência, cada posição do vetor de Contas possui um mutex próprio, denominado lock. Assim, sempre que a thread for tentar atualizar a o saldo de uma conta, trava-se o lock específico da conta, realiza-se a operação e destrava o lock.

5. Observações

Na função que realiza as transferências, a thread primeiramente retira o elemento da fila, e, após isolá-lo, realiza a operação de transferência.

Durante a execução das threads, se a fila estiver vazia e a *flagFim* for igual a 1, destrava-se o mutex (global), aumenta-se o semáforo semaphore e quebra o loop do while, assim a thread finaliza sua execução e possibilita que as outras threads façam o mesmo.