Laboratório 5
Implementação dos padrões leitores/escritores e barreira

Programação Concorrente (ICP-361)
Prof. Silvana Rossetto

[1]IC/CCMN/UFRJ

Introdução

O objetivo deste Laboratório[1] é praticar a implementação de padrões clássicos de programação concorrente usando locks e variáveis de condição.

Atividade 1

Objetivo: Mostrar um programa concorrente que faz acesso a uma **lista de inteiros ordenada** compartilhada.

Descrição: As operações sobre a lista são: consulta (leitura), inserção (escrita) e remoção (escrita). As operações de consulta podem ser feitas ao mesmo tempo por diferentes threads. As operações de escrita (inserção e remoção) devem ser feitas de forma atômica/exclusiva pois alteram a lista.

Pode ocorrer, por exemplo, de uma thread consultar um valor enquanto outra thread o exclui. A primeira thread pode reportar que o valor está na lista quando de fato ele pode já ter sido excluído pela segunda thread antes da primeira thread retornar. Esse é um exemplo de situação em que uma thread está lendo enquanto outra está escrevendo (alterando) a estrutura de dados compartilhada.

Outra situação que pode ocorrer é uma thread buscar por um valor (10, por exemplo) enquanto outra thread exclui um valor que está em uma posição anterior na lista (7, por exemplo). A primeira thread pode ter passado pelo valor 6 (imediatamente anterior a 7 na lista) e armazenado o ponteiro para o próximo valor (no caso, o 7). Mas antes de acessá-lo, o elemento com o valor 7 é excluído. Dessa forma, a primeira thread poderá acessar uma área de memória inválida (ou realocada para outro programa).

Roteiro:

1. Abra os arquivos `list_int.h` e `list_int.c` para ver a implementação do TAD lista. Essa implementação foi extraída do livro *An Introduction to Parallel Programming*, Peter Pacheco, Morgan Kaufmann, 2011 (cap4). *Acompanhe a explanação da professora*.
2. O TAD lista implementado nesses arquivos poderia ser compartilhado por threads de um programa concorrente? Com qual finalidade e de que forma?
3. Abra o arquivo `main_lock.c` para ver um exemplo de uso compartilhado da lista encadeada em um programa concorrente.
4. O que poderia acontecer se o programa não implementasse exclusão mútua no acesso às operações da lista encadeada?

[1]Parcialmente extraído do livro *An Introduction to Parallel Programming*, Peter Pacheco, Morgan Kaufmann, 2011 (cap4)

5. Compile (inclua na linha de compilação os dois arquivos .c) e execute o programa variando o número de threads de 1 a 4.
6. O que acontece com o tempo de execução do programa quando aumentamos o número de threads? Por que isso ocorre?

Atividade 2

Objetivo: Mostrar o uso do lock especial (`rwlock`) de leitura e escrita.

Descrição: A biblioteca pthread oferece uma implementação de um *lock* **read_write**. Ele é como um *lock* de exclusão mútua, exceto que provê duas operações de *lock*: uma que aloca para leitura e outra que aloca para escrita. Várias threads podem obter o *lock* de leitura simultaneamente, enquanto apenas uma thread pode obter o *lock* de escrita de cada vez.

Roteiro:

1. Abra o arquivo `main_rwlock.c` e entenda como o *lock* **read_write** foi usado. *Acompanhe a explanação da professora*.
2. Compile e execute o programa.
3. Compare os tempos de execução com a versão anterior, que usava *locks* de exclusão mútua.
4. Em quais cenários o uso do **rwlock** pode ser mais vantajoso que o uso do *lock* de exclusão mútua?

Atividade 3

Objetivo: Experimentar o padrão de **sincronização coletiva** (barreira).

Roteiro:

1. Abra o arquivo `barreira.c`. Ele apresenta uma implementação do padrão barreira e um exemplo simples de uso. Acompanhe a explanação da professora.
2. Em que tipo de aplicação concorrente o padrão de sincronização coletiva com barreira precisa ser usado?
3. Execute o programa e verifique seus resultados.
4. Comente a linha 45 (que chama a 'barreira'). Execute novamente o programa e avalie os resultados.

Entrega do laboratório Responda as questões colocadas (em vermelho) em cada atividade.