Laboratório 6
Implementação de uma aplicação concorrente usando o padrão leitores/escritores

Programação Concorrente (ICP-361)
Profa. Silvana Rossetto

[1]IC/CCMN/UFRJ

Introdução

O objetivo deste Laboratório[1] é apresentar e usar o padrão de concorrência **leitores e escritores**.

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 da estrutura de dados **lista de inteiros**. Essa implementação foi extraída do livro *An Introduction to Parallel Programming*, Peter Pacheco, Morgan Kaufmann, 2011 (cap4). Acompanhe a explanação da professora.
2. A estrutura de dados lista implementada nesses arquivos pode ser compartilhada por threads de um programa concorrente? Com qual finalidade e de que forma?
3. Abra o arquivo `error_main.c` para ver um exemplo de uso compartilhado da lista encadeada em um programa concorrente.

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

4. O que poderá acontecer na execução do programa, dado que ele não implementa exclusão mútua no acesso às operações da lista encadeada?
5. Compile (`gcc error_main.c list_int.c -Wall`) e execute o programa variando o número de threads de 1 a 4.
6. Avalie os resultados.

Esse é um exemplo de uso de uma estrutura de dados que não é *thread-safe*, ou seja, não está preparada (internamente) para ser usada em um programa concorrente.

Atividade 2

Objetivo: Mostrar **como usar a estrutura de dados lista da Atividade 1 de forma correta** em um programa concorrente usando **exlusão mútua**.

Descrição: Vamos usar uma variável de *lock* para garantir exclusão mútua no acesso das operações de **leitura** e **escrita** na lista.

Roteiro:

1. Vamos manter os arquivos `list_int.h` e `list_int.c` como estão.
2. Abra o arquivo `main_lock.c` para ver um exemplo de **uso compartilhado e seguro** da lista encadeada em um programa concorrente.
3. Compile (`gcc main_lock.c list_int.c -Wall`) e execute o programa variando o número de threads de 1 a 4. O tempo medido está mensurando a sobrecarga da sincronização, espera-se que ele aumente com o aumento do número de threads! Por que?
4. Avalie os resultados.

Atividade 3

Objetivo: Mostrar o uso do lock especial (**rwlock**) que implementa o padrão de concorrência **leitores/escritores**.

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 (`gcc main_rw.c list_int.c -Wall`) e execute o programa variando o número de threads de 1 a 4.
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?

5. **Experimente ambas as versões da aplicação, variando seus parâmetros e comparando os resultados.**

Recompile os programas com a flag de otimização -O2 e -O3 e refaça os experimentos.