Laboratório 4
Comunicação entre threads via memória compartilhada

Programação Concorrente (ICP-361) 2024-1
Prof. Silvana Rossetto

[1]Instituto de Computação/UFRJ

Introdução

O objetivo deste Laboratório é introduzir o uso de variáveis compartilhadas (globais) para permitir a comunicação entre as threads de uma aplicação e mostrar quais benefícios e desafios essa comunicação pode trazer.

Para cada atividade, siga o roteiro proposto e responda às questões colocadas.

Atividade 1

Objetivo: Mostrar um exemplo simples de programa com uma variável compartilhada entre threads, e os possíveis efeitos indesejáveis do acesso compartilhado.

Roteiro:

1. Abra o arquivo **exemplo1.c** e entenda o que ele faz. **Qual saída é esperada para o programa (valor final da variável soma)?**
2. Execute o programa **várias vezes** e observe os resultados impressos na tela (acompanhe a explanação da professora).
3. Os valores impressos foram sempre o valor esperado? Se não, explique por que isso aconteceu?
4. É possível que a variável **soma** termine com valor acima de 200000? Por que?

Relatório da atividade: Em um arquivo de texto, inclua as respostas para as questões em vermelho colocadas acima.

Atividade 2

Objetivo: Introduzir o uso de *locks* provido pela biblioteca Pthreads.

Roteiro:

1. Abra o arquivo **exemplo2.c** e compreenda como locks são usados para implementar a **exclusão mútua** entre as threads (acompanhe a explanação da professora).
2. Execute o programa **várias vezes**. Os valores impressos foram sempre o valor esperado? Por que?
3. Altere o número de threads e avalie os resultados.

Relatório da atividade: No mesmo arquivo de texto, acrescente as respostas para as questões em vermelho colocadas acima.

Atividade 3

Objetivo: Praticar o uso da concorrência. Dada uma sequência consecutiva de números naturais (inteiros positivos) de 1 a N (N muito grande), identificar todos os **números primos** dessa sequência e retornar a **quantidade total de números primos encontrados**. Use a função abaixo para verificar a primalidade de um número:

```
int ehPrimo(long long int n) {
   int i;
      if (n<=1) return 0;
      if (n==2) return 1;
      if (n%2==0) return 0;
      for (i=3; i<sqrt(n)+1; i+=2)
         if(n%i==0) return 0;
      return 1;
```

Roteiro

1. Implemente uma **versão concorrente** para esse problema, dividindo a tarefa entre as threads. O número de elementos (N) e o número de threads devem ser informados na linha de comando.
   IMPORTANTE: Use o padrão de **bolsa de tarefa** para distribuição da tarefa entre as threads: cada thread pega o próximo número da série para avaliar a sua primalidade (veja Aula 4, de 16 de abril).
   ATENÇÃO: os valores da série NÃO devem ser colocados em um vetor, devem ser obtidos incrementalmente em tempo de execução.
2. OBS.: defina a variável N do tipo **long long int** e use a função **atoll()** para converter o valor recebido do usuário (string) para long long int.
3. Certifique-se que a sua solução está correta.
4. Inclua tomadas de tempo no código (parte de processamento da sequência).
5. Execute o programa várias vezes, variando o valor de N (experimente 10^3 e 10^6); e para cada valor de N, varie a quantidade de threads (1, 2 e 4).
6. Para cada configuração, execute o programa ao menos 5 vezes e registre o tempo médio dessas execuções.
7. Calcule a **aceleração e a eficiência** alcançada em cada configuração, tomado como base (execução sequencial) a execução com 1 thread.

Relatório da atividade: No mesmo arquivo de texto, descreva as medidas obtidas (tempo médio, aceleração e eficiência) para cada configuração de teste (variando N e o número de threads, como solicitado acima). Acrescente a sua análise sobre os resultados obtidos.

Entrega do laboratório: Esse Lab não tem entrega. Traga as anotações completas feitas no arquivo texto na próxima aula de laboratório, dia 25 de abril.