Laboratório 5
Comunicação entre threads via memória compartilhada

Computação Concorrente (MAB-117)
Prof. Silvana Rossetto

[1]DCC/IM/UFRJ — 29 de setembro de 2016

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 problemas essa comunicação pode gerar.

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

Atividade 1

Objetivo: Mostrar o comportamento **não-determinístico** de um programa com três threads, ilustrando uma situação de **condição de corrida**.

Roteiro:

1. Abra o arquivo **exemplo1.c** e entenda o que o programa faz. **Quais são os possíveis valores finais da variável `y` e por que? É possível `y` terminar com valor 3 (combinação binária dos números 1 e 2)? Por que?**
2. Compile e execute o programa **várias vezes** e observe os resultados impressos na tela.
3. Altere o código do programa modificando a ordem em que as threads são criadas e repita as execuções.
4. Esse programa apresenta o problema de **condição de corrida**? Se sim, ele é tolerável ("condição de corrida não-ruim") nessa aplicação?

Atividade 2

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 **exemplo2.c** e entenda o que ele faz. **Qual saída é esperada para o programa (valor final da variável s)?**
2. Compile e execute o programa **várias vezes** e observe os resultados impressos na tela.
3. **Os valores impressos foram sempre o valor esperado? Se não, explique por que isso aconteceu?**
4. Esse programa apresenta o problema de **condição de corrida**? Se sim, ele é tolerável ("condição de corrida não-ruim") nessa aplicação?

Atividade 3

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

Roteiro:

1. Abra o arquivo **exemplo3.c** e compreenda como locks são usados para implementar a exclusão mútua (*acompanhe a explanação da professora*).
2. Execute o programa **várias vezes**. Os valores impressos foram sempre o valor esperado?
3. Altere o número de threads e avalie os resultados.

Atividade 4

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

Roteiro: Dada uma sequência de números inteiros positivos 1 a N (N muito grande), identificar todos os **números primos** 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;
```

1. Implemente uma **versão sequencial** e certifique-se que está correta. O número de elementos (N) deve ser informado na linha de comando. Inclua tomadas de tempo no código.
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. Implemente uma **versão concorrente** onde cada thread pega o próximo número da série para avaliar a sua primalidade (veja Aula 5). **Certifique-se que a sua solução está correta (compare os resultados com a versão sequencial)**. O número de elementos (N) e o número de threads devem ser informados na linha de comando. Inclua tomadas de tempo no código.
4. Calcule o ganho de desempenho da versão concorrente.

Empacote e envie seu programa e o arquivo lab6.txt para correção Crie um diretório e o nomeie juntando seu "primeiro" e "último" nome. Copie pra dentro desse diretório o código fonte do programa da atividade 4 (versões sequencial e paralela). Comprima o diretório (ex., `zip -r JoseSilva.zip JoseSilva/`) e envie o arquivo comprimido para o endereço de email **computacao.concorrente.ufrj@gmail.com** com subject "**CompConc Lab5**".

O email deve ser enviado até amanhã, dia 30/09.