Modulo 1 - Lista de Exercícios (2020/1 REMOTO)

Computação Concorrente (MAB-117)

Prof. Silvana Rossetto

DCC/IM/UFRJ

23 de dezembro de 2020

Questão 1) Considere o programa mostrado abaixo, que contabiliza a quantidade de números negativos presentes em um vetor de inteiros e responda as questões colocadas:

(a) Esse programa está correto (i.e., ele calcula corretamente a quantidade de valores negativos no vetor)?
R: Sim, o programa aborda o problema quebrando o vetor em blocos e cada thread varre os elementos alocados a ela e testando se são menores que 0.

(b) E possível executar o programa com sucesso passando um número qualquer de threads na linha de comando?
R: Exceto no caso onde é passado "0", o programa irá executar corretamente mesmo que o número de threads seja maior que o de elementos.

(c) A carga de trabalho será sempre balanceada entre as threads, independente do número de threads informado?
R: Quando houver resto na divisão do vetor pelo número de threads a última terá carga maior.

(d) Há condição de corrida nesse código?
R: Como as threads acessam posições de memória distintas tanto na leitura quanto na escrita, a ordem de execução não irá alterar o resultado final e, consequentemente, não há condição de corrida.

Questão 2) Responda as questões abaixo:

(a) O que caracteriza que um programa é concorrente e não sequencial?
R: Um programa concorrente contém mais de um contexto de execução ativo ao mesmo tempo, onde os mesmos podem ser executados simultaneamente.

(b) O que é seção crítica do código?
R: É uma seção do código onde uma posição de memória é compartilhada entre as threads e que não deve ser acessada concorrentemente.

(c) O que significa uma operação ser atômica?
R: Significa que uma vez iniciada só permitirá a execução por outra thread quando executada por completo.

(d) Como funciona a sincronização por exclusão mútua?
R: A seção crítica é executada como operação atômica, impedindo sua execução concorrente. Dada uma thread A que inicia a execução da seção crítica, nenhuma outra poderá iniciar sua execução até que A tenha terminado (inclusive se A perder a CPU antes de executar a seção de saída, o trecho permanece bloqueado para as outras threads).

Questão 3) Uma aplicação dispara três threads (T1, T2 e T3) para execução (códigos mostrados abaixo).
(a) Verifique se os valores −3, −2, 0, 2, 3 podem ser impressos na saída padrão quando essa aplicação é executada. Em caso afirmativo, mostre uma sequência de execução das threads que gere o valor correspondente.

```
int x=0; //variável global

        T1:                 T2:             T3:
(1)     x++;                x--;            x--;
(2)     x--;                x++;            x++;
(3)     x++;                                x--;
(4)     if (x == 1)                         if(x == -1)
(5)         printf("%d",x);                     printf("%d",x);
(6)
```

R: -3 -> T3 executa apenas a leitura de x em (1) [x==0] -> T1 executa (1), (2), (3), (4) [x==1 e entra no if] -> T3 termina de executar (1) com valor lido inicialmente -> T2 lê x [x==-1] -> T3 executa (2) -> T2 termina a execução de (1) com o x lido [x==-2] -> T3 executa (3) [x==-3] -> T1 executa (5) [x==-3] -> Programa imprime -3

Aluno: Lucas de Queiroz Silva e Silva
DRE: 115197960

-2 -> T3 executa apenas a leitura de x em (1) [x==0] -> T1 executa (1), (2), (3), (4) [x==1 e entra no if] -> T3 termina de executar (1) [x==-1] -> T3 executa (2) [x==0] -> T2 executa (1) [x==-1] -> T3 executa (3) [x==-2] -> T1 executa (5) [x==-2] -> Programa imprime -2

0 -> T3 executa apenas a leitura de x em (1) [x==0] -> T1 executa (1), (2), (3), (4) [x==1 e entra no if] -> T3 termina de executar (1) [x==-1] -> T3 executa (2) [x==0] -> T1 executa (5) [x==0] -> Programa imprime 0

2 -> T1, T2 e T3 realizam leitura simultânea de x==0 -> T3 executa (1), (2), (3), (4) [x==-1 e entra no if] -> T2 executa (1) -> T1 executa (1) [x==1] -> T2 executa (2) [x==2] -> T3 executa (5) [x==2] -> Programa imprime 2

3 -> T1, T2 e T3 realizam leitura simultânea de x==0 -> T3 executa (1), (2), (3), (4) [x==-1 e entra no if] -> T2 executa (1) -> T1 executa (1) [x==1] -> T1 e T2 realizam leitura simultânea de x em (2) -> T1 executa (2) -> T2 executa 2 [x==2] -> T1 executa 3 [x==3] -> T3 executa (5) [x==3] -> Programa imprime 3

Questão 4) O código abaixo apresenta uma proposta de implementação de exclusão mútua com espera ocupada (ao invés de serem bloqueadas, as threads executam um loop de entrada na seção crítica). A solução proposta prevê apenas duas threads (T0 e T1). (a) Essa implementação garante exclusão mútua? Se sim, argumente justificando sua resposta. Se não, descreva cenários de execução que mostrem que a solução é incorreta.

```
boolean queroEntrar_0 = false, queroEntrar_1 = false;
```

T0

```
while(true) {
(1) while(queroEntrar_1) { ; }
(2) queroEntrar_0 = true;
(3) //executa a seção crítica
(4) queroEntrar_0 = false;
(5) //executa fora da seção crítica
}
```

T1

```
while(true) {
(1) while(queroEntrar_0) { ; }
(2) queroEntrar_1 = true;
(3) //executa a seção crítica
(4) queroEntrar_1 = false;
(5) //executa fora da seção crítica
}
```

R: A implementação não garante exclusão mútua pois caso as threads executem as comparações antes do valor de queroEntrar_X ser atualizado, ambas ganham acesso a seção crítica antes dela ser bloqueada.
Exemplo: T0 executa (1) [sai da fila de espera] -> T1 executa (1) [sai da fila de espera] -> A partir daqui as threads já passaram da barreira e podem executar a seção crítica de forma concorrente.

Questão 5) O código abaixo apresenta outra proposta de implementação de exclusão mútua com espera ocupada. A solução proposta prevê apenas duas threads (T0 e T1). (a) Essa implementação garante exclusão mútua? Se sim, argumente justificando sua resposta. Se não, descreva cenários de execução que mostrem que a solução é incorreta.

```
boolean queroEntrar_0 = false, queroEntrar_1 = false; int TURN;
```

T0

```
while(true) {
(1) queroEntrar_0 = true;
(2) TURN = 1;
(3) while(queroEntrar_1 &&
          TURN == 1) { ; }
(4) //executa a seção crítica
(5) queroEntrar_0 = false;
(6) //executa fora da seção crítica
}
```

T1

```
while(true) {
(1) queroEntrar_1 = true;
(2) TURN = 0;
(3) while(queroEntrar_0 &&
          TURN == 0) { ; }
(4) //executa a seção crítica
(5) queroEntrar_1 = false;
(6) //executa fora da seção crítica
}
```

R: Garante pois a inclusão da variável "TURN" e da atualização do valor de "queroEntrar_X" antes da execução do loop fazem com que caso uma thread chegue no passo 3 ela necessariamente bloqueia a execução da outra por "queroEntrar_X" e o fluxo de execução da própria thread é bloqueado por "TURN", fazendo com que caso uma thread entre na seção crítica a segunda fique em loop de noop sempre que assumir a CPU até a primeira atualizar o valor de "queroEntrar_X".

Aluno: Lucas de Queiroz Silva e Silva
DRE: 115197960