Mini EP4 - Problemas de Concorrência

Lucas de Sousa Rosa

31 de agosto de 2024

Motivação

Em algumas versões antigas do Java[1], valores de 64 bits (como `long` e `double`) eram tratados de forma peculiar: uma operação de escrita ou leitura nesses tipos era realizada como duas operações separadas de 32 bits. Isso podia levar a problemas de concorrência em ambientes multithread, onde uma thread poderia ler um valor "meio atualizado", misturando partes do valor antigo e do novo.

Podemos simular esse comportamento em C usando uma variável global compartilhada de 64 bits (`uint64_t`) e duas threads: uma que escreve um novo valor na variável (em duas etapas de 32 bits) e outra que lê o valor. O arquivo `long-problem.c` contém a implementação em questão.

A leitura da variável compartilhada pode acontecer em três momentos distintos: após as duas escritas, entre as duas escritas, ou antes das duas escritas. Dos três momentos, a leitura entre as duas escritas representa uma história incorreta; é estranho pensar em uma "meia atualização". O valor lido da variável pode assumir três valores possíveis:

- Caso a leitura tenha acontecido após as duas escritas, o valor da variável será 0x8765432112345678 (história correta).
- Caso a leitura tenha acontecido entre as duas escritas, o valor da variável será 0x12345678 (história incorreta). Os outros 32 bits são interpretados como "lixo".
- Caso a leitura tenha acontecido antes das duas escritas, o valor da variável será 0x0 (história correta).

Os números estão sendo representados em hexadecimal (prefixo `0x`) para facilitar a visualização. Além disso, o código fornecido tem um detalhe: ele força a ocorrência da história incorreta.

Uma forma de resolver esse problema seria tornando a escrita uma operação atômica, que no nosso código poderia ser feito através de um mutex. O arquivo `long-problem-fixed.c` adiciona um mutex para controlar as leituras e escritas.

O script `run.sh` compila ambos os fontes e executa-os 100 vezes, contando o número de vezes em que cada história aconteceu. Você pode executá-lo com o comando `./run.sh`. Na execução abaixo, podemos observar que o código original resultou a maior parte das vezes em histórias incorretas (78%). Enquanto que na versão corrigida nenhuma história foi incorreta (0%).

```
lucas@kamaji:~/ws/monitoria-PPD$ ./run.sh
make: Nada a ser feito para 'all'.
Resultados do código original (com problema de concorrência):
Número de ocorrências de 0x12345678: 78
Número de ocorrências de 0x0: 20
Número de ocorrências de 0x8765432112345678: 2

Resultados do código corrigido (com mutex):
```

[1] `https://docs.oracle.com/javase/specs/jls/se7/html/jls-17.html#jls-17.7`

```
Número de ocorrências de 0x12345678: 0
Número de ocorrências de 0x0: 30
Número de ocorrências de 0x8765432112345678: 70
```

Todos os arquivos mencionados podem ser encontrados no arquivo `mini_ep4.zip` presente no e-Disciplinas.

Sua Tarefa

Procure outro exemplo de código (em C ou outras linguagens) que pode levar a problemas de concorrência devido à falta de atomicidade em operações ou acessos a dados compartilhados. Forneça o código (ou pseudocódigo) e explique brevemente o problema escolhido e como ele poderia ser evitado. Você deve entregar um arquivo PDF ou TXT com a explicação. Você pode entregar o código-fonte e arquivos auxiliares separadamente em um arquivo compactado.